НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-додаток для підтримки процесу сервісного обслуговування автомобілів»

Виконав:
студент IV курсу, групи КП-61
Телефус Ілля Анатолійович _____

Керівник:
Старший викладач кафедри ПЗКС,
Гадиняк Руслан Анатолійович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Старший викладач кафедри ПМА,
Мальчиков Володимир Вікторович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Телефусу Іллі Анатолійовичу

1. Тема проєкту «Веб-додаток для підтримки процесу технічного обслуговування автомобілів», керівник проєкту Гадиняк Руслан Анатолійович, старший викладач, затверджені наказом по університету від «___»_____ 2020 р. №_____

2. Термін подання студентом проєкту «15» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз мов програмування та технологій розроблення web-сайтів;
   - розроблення web-ресурсу центру електронної освіти;
   - опис розроблених алгоритмів та підпрограм;
   - аналіз розробленого web-ресурсу.

5. Перелік обов'язкового графічного матеріалу:
   - структура бази даних (креслення);
   - діаграма прецедентів (креслення);

- структурна схема системи (плакат);
- аналіз розміру скомпільованих компонентів (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	16.11.2019	
2.	Розроблення та узгодження технічного завдання	09.12.2019	
3.	Розроблення структури web-ресурсу	30.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	16.01.2020	
5.	Розроблення дизайну сторінок та графічних елементів	25.01.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	30.01.2020	
7.	Програмна реалізація web-ресурсу	05.02.2020	
8.	Тестування web-ресурсу	20.02.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	20.03.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	10.04.2020	
11.	Підготовка графічної частини дипломного проєкту	19.05.2020	
12.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Ілля ТЕЛЕФУС

Керівник проєкту Руслан ГАДИНЯК

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню веб-додатку для підтримки процесу технічного обслуговування автомобілів.

Розроблене програмне забезпечення є односторінковим веб-додатком, що призначений для спрощення запису на ТО та планові роботи, оптимізації комунікації між сервісними центрами та власниками транспортних засобів та автоматизації повного циклу проходження регламентних робіт.

У дипломному проєкті здійснено глибокий аналіз наявних веб-сервісів та сайтів, що здійснюють сервісний огляд та регламентні роботи. Були поставлені функціональні та нефункціональні вимоги до проєкту розробленого веб-додатку, продумана бізнес-логіка та визначено основні ролі, відповідно яким реалізовано необхідну функціональність.

У веб-додатку передбачена реєстрація нових користувачів за допомогою форм, а також авторизація через системи компаній-партнерів. Кожна сторінка є захищеною та недоступною не автентифікованим користувачам. Серверну частину розбито на функціональні модулі, що дає можливість створення інтеграцій із компаніями, що мають великі автопарки.

У даному дипломному проєкті розроблено: REST API модуль для комунікації серверної та клієнтської частин, модуль інтеграції із Unity CRM, модуль управління базою даних та поштовий модуль. Реалізовано зрозумілий та інтуїтивний UX, адаптивний уніфікований дизайн веб-сторінок, інтеграцію із сервісами Stripe для проведення оплати та Google Maps для використання мап. Веб-додаток підтримує можливість інтеграції з використанням REST API, що надається компаніям-партнерами.

ABSTRACT

This diploma project deals with the creation of a web application for car maintenance support.

Web service is developed as a single page web application designed for simplifying car maintenance, MOT, servicing and diagnostics as well as for optimizing communication between service centers, garages and vehicle owners to automate the full cycle of routine work.

The diploma project provides an in-depth analysis of existing web services and sites that provide servicing and MOT. All the functional and non-functional requirements were set for the developed web service. All the roles were defined for the great user experience and high efficiency.

The web application provides a registration page, as well as authorization using the partner companies' systems. Each page is protected and inaccessible to non-authenticated users. The server side is divided into modules, which makes it possible to create integrations with companies that own large fleets.

This project led to developing a bunch of modules, services and integrations, such as: REST API module for communication of server and client parts, Unity CRM integration module, database management module and mail module. Clear and intuitive UX was implemented alongside with adaptive unified web page design, Stripe integration for payment and Google Maps for showing all the garages on the map inside the application. The web application gives the possibility of integration using the REST API provided by partner companies.

ДП.045440-01-90 Веб-додаток для підтримки процесу технічного обслуговування автомобілів. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-додаток для	5	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Технічне завдання		
ДП.045440-03-81	Веб-додаток для	60	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-додаток для	4	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-додаток для	8	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-додаток для	1	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Функціональність		
	програмних засобів. UML		
	діаграма прецедентів		
ДП.045440-07-99	Веб-додаток для	1	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Структура бази даних.		
	ER діаграма		
ДП.045440-08-98	Веб-додаток для	1	
	підтримки процесу		
	процесу технічного		
	обслуговування		
	автомобілів.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ілля ТЕЛЕФУС

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: веб-додаток для підтримки процесу технічного обслуговування автомобілів.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання я якості веб-додатку, що призначений для підтримки процесу технічного обслуговування автомобілів. Може працювати у вигляді самостійного продукту, та для інтеграції з API компаній, що мають великі автопарки.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-сервіс має забезпечувати такі основні функції:

1) можливість постановки автомобілів для технічного обслуговування;
2) можливість пропонувати власникам автомобілів необхідні та планові процедури зважаючи на інформацію про авто;
3) підбір станцій ТО, що знаходяться поряд з користувачем та наявність там вільних місць;
4) можливість розширення за рахунок інтеграції з компаніями, що мають великі автопарки та хочуть автоматизувати процеси з ТО автомобілів;

5) можливість перегляду бронювань та статусу бронювань ТО;

6) здійснення оплати онлайн;

7) реєстрація користувачів у системі за допомогою профілів у системах компаній де вони працюють, а також створення нових профілів на основі введених даних.

Розробку серверної частини виконати мовою програмування Java, використовуючи серверний фреймворк Spring. Розроблення клієнтської частини виконати, веб-фреймворк для створення односторінкового веб-додатку Angular8.

Додаткові вимоги:

1) завантаження звітів про роботу та про стан автомобіля зі сторінки бронювань;

2) наявність адаптивного дизайну для двох платформ: настільних комп'ютерів та мобільних пристроїв;

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

- «Структура бази даних. ER діаграма»;
- «Функціональність програмних засобів. UML діаграма прецедентів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту 16.11.2019

Розроблення та узгодження технічного завдання..... 09.12.2019

Підготовка матеріалів першого розділу дипломного проєкту 30.12.2019

Розроблення структури web-додатку 16.01.2020

Підготовка матеріалів другого розділу дипломного проєкту..... 30.01.2020

Створення структури бази даних..... 10.01.2020

Заповнення бази даних через інтеграцію зі стороннім API..... 20.01.2020

Програмна реалізація і тестування web-додатку 20.02.2020

Підготовка третього розділу дипломного проєкту..... 20.03.2020

Підготовка четвертого розділу дипломного проєкту 10.04.2020

Підготовка графічної частини дипломного проєкту 19.05.2020

Оформлення документації дипломного проєкту 26.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ілля ТЕЛЕФУС

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Front-end − це інтерфейс для взаємодії між користувачем і back end.

Back-end – серверна частина.

ПЗ – програмне забезпечення.

БД – база даних.

UI – інтерфейс користувача.

СТО – станція технічного обслуговування.

VIN – це унікальний серійний номер, що застосовується в автомобільній промисловості для індивідуального розпізнавання кожного механічного транспортного засобу.

SPA – односторінковий веб-застосунок.

SSR – серверний рендеринг.

MPA – багатосторінковий веб-застосунок.

CLI – консольний інтерфейс.

UX – досвід користування.

CRM – управління взаємовідносинами зі споживачами.

CI – процес неперервної інтеграції та постійного запуску тестування.

CD – безперервна доставка, постійна розробка нової функціональності.

ВСТУП

Основна мета дипломного проєкту – розробка веб-додатку для підтримки процесу технічного обслуговування автомобілів, що дозволяє власникам транспортних засобів не чекати в чергах на СТО, здійснювати моніторинг технічного стану автомобілів, отримувати сповіщення щодо статусу виконаних робіт та рекомендації щодо замовлення відповідних видів обслуговування. Підставою для розробки даного продукту стала проблема високої завантаженості станцій технічного обслуговування автомобілів та відгуки водіїв, що марнують свій час в чергах, адже не мають інформації про наявність вільних місць.

Станом на початок 2016 року кожен п’ятий житель України має хоч одне авто [1]. Володіння транспортним засобом передбачає регламентні роботи, які кожен власник має проводити з певною періодичністю або за настання потреби. На жаль, не всі водії можуть проводити технічний огляд автомобіля повною мірою, а також виконувати ремонт, тому переважна більшість звертається до СТО. Незалежно від того, чи це сервіс офіційного дилера, чи приватної особи – водіям доводиться мати справу з чергами, відсутністю вільних місць та/чи деталей для виконання відповідних робіт.

Також у водіїв часто виникають питання, скільки коштує та чи інша деталь, яка ціна роботи спеціаліста наскільки швидко буде проведений ремонт.

Не менш актуальною проблемою є безпосередня комунікація замовника та виконавця, що буває не дуже продуктивною і для того, щоб звести її до мінімуму необхідно надати користувачу можливість відслідковувати стан виконання робіт у режимі онлайн.

Для вирішення всіх описаних проблем можна використати веб-застосунок, що дозволить зареєструвати авто в системі, автоматично підбирати необхідні серісні роботи, переглядати статус замовлень, отримувати сповіщення на електронну пошту та, крім цього, матиме

надзвичайно простий користувацький інтерфейс, який буде інтуїтивно допомогати користувачу полегшити процес технічного обслуговування автомобілів.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1. Загальний опис проблеми процесу підтримки технічного огляду автомобілів

Технічне обслуговування автомобілів – доволі складний і багатогранний процес, оскільки під час регламентних робіт досить часто виявляють інші проблеми, що потребують втручання. У наш час кожен власник транспортного засобу має обов'язково здійснювати ТО аби не наражати на небезпеку себе та оточення. Враховуючи сьогоденні реалії більшість водіїв користуються послугами приватних організацій, майстрів та СТО, хоча є і ті, хто виконують ремонт авто лише в офіційного дилера. Проте, як у випадку з неофіційним сервісом, так і з офіційним, організацій, що надають доступ до систем, де автомобілісти, зареєструвавшись, можуть здійснювати постановку транспортних засобів на проходження ТО в режимі онлайн одиниці.

Також варто зазначити, що рішення, що вже існують на ринку, мають досить малу функціональність і, здебільшого, мають поганий інтерфейс, що навпаки відштовхує потенційну цільову аудиторію. Серед альтернатив у нашій країні є декілька сервісів, що мають більш продуманий інтерфейс та ряд додаткових функцій, проте все одно є досить складними та не продуманими.

Загалом завданнями веб-застосунку для автоматизації проходження ТО є забезпечення постановки авто на ТО, здійснення сервісних робіт, аналіз стану авто та його характеристик для здійснення оптимального обслуговування а також інформування власника протягом усього процесу від моменту, коли автомобіль надходить майстру, до здачі відремонтованого транспортного засобу господарю.

В цілому, даний вид програмного забезпечення був створений досить нещодавно, адже ще 5-10 років назад ніхто і не замислювався над

автоматизацією даного процесу, оскільки досить мало людей мали транспортні засоби.

У найперших таких веб-додатках функціональність була реалізована у вигляді статичної сторінки та форми зв'язку з можливістю ввести номер телефону. Цілком зрозуміло, що автоматизувати процес через запис за допомогою дзвінків дуже важко, бо потрібні люди, що будуть фіксувати замовлення, відповідати клієнтам, скасовувати записи і т.д. Очевидно, що такими веб-додатками водії не користувалися або через незручність, або через звичку їздити на знайомі СТО.

Важливу роль у розвитку платформ підтримки процесу технічного обслуговування автомобілів відіграли два чинники – збільшення попиту за рахунок зростання кількості авто та розвиток веб технологій, UX та маркетингу. Дані продукти стали рекламуватися, про них стало відомо більшій кількості людей, з'явилася можливість вибрати день та час для виконання сервісних робіт, зареєструвати авто для проходження регламентного огляду, інколи такі веб-додатки навіть показували наявність тих чи інших деталей у сервісних центрах.

Головна складова веб-додатку для підтримки процесу технічного обслуговування автомобілів – відсутність людського чинника в більшості процесів. Наприклад, при телефонній розмові існує ризик, що оператор допустить помилку чи неправильно зрозуміє, які роботи хоче замовити водій. Також, щоб відмінити бронювання, клієнт має телефонувати менеджеру. У свою чергу, сервісні станції повинні бути проінформовані, майстри мають надати приблизну оцінку, скільки часу потребують ті чи інші види робіт.

В результаті автоматизації цих основних та багатьох інших факторів витрати на людські ресурси зменшуються, вірогідність помилки прямує до нуля. Також збільшується швидкодія системи та її стабільність, адже, як правило, на сервері присутні резервні копії всіх даних, і навіть при виникненні збоїв веб-застосунок залишається функціональним.

Основні переваги веб-додатку для підтримки процесу технічного обслуговування автомобілів:

1. Зрозумілість та сценарій, який дозволяє послідовно вибрати усі пункти, необхідні для заявки на проходження ТО.
2. Можливість обрати вид сервісу.
3. Мінімізація скасувань за рахунок того, що користувач сам обирає необхідні опції, а також може скасувати бронювання повернувшись на головну з будь-якого етапу до виконання оплати.
4. Зручність – всі операції проходять онлайн, достатньо лише зареєструватись чи увійти в систему після надання персональних даних, витративши усього пару хвилин.
5. Прозорість – клієнт бачить, яку суму доведеться заплатити за послуги відразу, без прихованих платежів.
6. Платформа автоматично генерує договір по наданню послуг.
7. Користувач може бачити статус замовлення.
8. Свобода вибору – користувач бачить СТО поблизу та може обрати будь-яку, не перебуваючи під впливом реклами.
9. Власник авто може залишити відгуки.
10. Безпека – уся приватна інформація захищена, а також використовується шифрування при виконанні транзакцій.
11. Зручний UI.

Врахувавши всі переваги веб-додатку підтримки процесу технічного обслуговування автомобілів, можна зробити висновок, що це актуальне рішення такої багатогранної проблеми.

1.2. Аналіз існуючих рішень для даної задачі

Проаналізувавши існуючі прототипи та закінчені продукти, що використовуються для вирішення даної проблеми, стає цілком очевидно, що кожен з них має в собі певні недоліки. Для того, щоб розробити ПЗ, що буде користуватися попитом, необхідно витратити чимало часу та витрати

чималі фінансові ресурси. Тому, нижче наведені приклади найбільш популярних та успішних веб-додатків для підтримки процесу технічного обслуговування автомобілів, а також їх переваги та недоліки.

Платформа My Eurocar (Skodakiev)

My eurocar – одна з найпопулярніших платформ в Україні та в Києві в цілому для запису та проходження ТО автомобіля та інших видів сервісних робіт. Цільова аудиторія даної платформи – водії, власники легкових транспортних засобів різних марок, які не хочуть чекати у чергах, а також обрати весь спектр послуг орієнтуючись на особисті уподобання. Основна задача платформи – створити зв'язок між клієнтами та станціями технічного обслуговування або сервісними центрами, та забезпечити їм комфортні умови співпраці і гарантувати відповідальність з боку зацікавлених сторін і безпеку укладання угод. Даний ресурс надає можливість власникам авто сформувати заявку для виконання сервісних робіт, у якій вони можуть обрати необхідні послуги, замовити діагностику та ряд інших робіт. Загалом платформа має зручний інтерфейс користувача, але через низьку швидкодію сервера в клієнтській частині, при переході з однієї сторінки на іншу, спостерігається значна затримка, особливо при низькій якості з'єднання із мережею. Також у клієнта немає можливості оформити заявку без попередньої реєстрації в системі та без указання VIN коду авто.

Переваги ПЗ:

1. Інтеграція з офіційним дилером Skoda.
2. Можливість обрати сервісні роботи власноруч.
3. Зручний інтерфейс користувача.
4. SPA підхід в клієнтській частині.
5. Наявність всієї необхідної інформації про послуги.
6. Прозорість (адже сервіс працює з авторизованими СТО).
7. Наявність декількох мов на платформі.

Недоліки ПЗ:

1. Низька швидкодія сервера, що призводить до затримки та довгого завантаження клієнтської частини.
2. Не можливо сформувати заявку на проходження ТО без попередньої реєстрації VIN номера авто в системі.
3. Висока кількість полів для заповнення, що ускладнює процес формування заявки на мобільних пристроях.
4. Не можна переглянути список СТО поруч, якщо у них немає місць на обрану дату.

Веб-застосунок Oiler

Oiler – другий за популярністю в Україні після My eurocar веб-застосунок для підтримки процесу технічного обслуговування автомобілів. Основне його завдання – автоматизувати роботу клієнтів і станцій технічного обслуговування та дозволити виконувати замовлення онлайн. Веб-застосунок надає можливість вибору потрібних діагностичних та ремонтних робіт і дозволяє обирати, деталі, які будуть встановлені в автомобілі замість зношених або зламаних. Також є можливість зв’язатися із менеджером для уточнення будь-яких деталей.

Переваги ПЗ:

1. Широкий вибір деталей, що будуть встановлені на авто.
2. Можливість замовити дзвінок оператора для уточнення деталей.
3. Зручний інтерфейс користувача.
4. Наявність багатьох додаткових можливостей у порівнянні з P2P платформами.
5. Наявність декількох мов на платформі.

Недоліки ПЗ:

1. Досить незрозумілий UX.
2. Низька швидкість завантаження клієнтської частини.
3. Відсутність SPA підходу в архітектурі.
4. Інтерфейс погано адаптований для мобільних пристроїв.

Веб-сайт Oil-service

Oil-service – веб-сайт, що орієнтований здебільшого на український ринок. Oil-service спеціалізується на наданні інформації про певні види сервісних робіт, їх вартість та приблизні строки. Також за допомогою інтерактивного блоку дозволяє розрахувати вартість повного комплекту регламентних робіт.

Переваги ПЗ:

1. Наявність інтерактивного модуля, що дозволяє порахувати приблизну вартість сервісних робіт.
2. Наявність відгуків на сайті.
3. Приємний дизайн.

Недоліки ПЗ:

1. Відсутність можливості оформити додаткові послуги, оскільки відсутній їх опис.
2. Відсутність SPA підходу в клієнтській частині.
3. Відсутність динамічного оновлення даних.
4. Висока кількість спливаючих вікон заважає переглядати контент.
5. Не надто зручний інтерфейс користувача.

Веб-додаток Autobooking

Даний ресурс є одним із найпопулярніших у Києві, адже він є свого роду агрегатором усіх наявних СТО та сервісів, що надають відповідні послуги для автомобілів. Використовуючи Autobooking власник транспортного засобу може обрати вид робіт, що необхідні, адресу, поблизу якої шукати СТО, марку автомобіля та день і час, що є зручним. Проаналізувавши дані, введені користувачем, веб-додаток відображає дві колонки, у одній з яких список СТО з інформацією, а в іншій – карта, де розміщені маркери з їх місцем розташування.

Таким чином, користувач має можливість обрати найкращий варіант по співвідношенню ціна-розташування. Також Autobooking дозволяє оформити заявку на проходження сервісних робіт та ТО онлайн.

Переваги ПЗ:

1. Приємний дизайн.
2. Наявність відгуків на сайті.
3. Велика кількість СТО.
4. Можливість бачити лише СТО, що обслуговують задану модель автомобіля.
5. Наявність декількох мов інтерфейсу.

Недоліки ПЗ:

1. Погана адаптованість для мобільних пристроїв.
2. При вході в режим відображення карти з мобільного пристрою його неможливо закрити і повернутися на список СТО.
3. Немає можливості, щоб додати декілька автомобілів в одному профілі користувача.
4. Не працює опція залишити відгук.

1.3. Постановка задачі

Результатом проведеного дослідження є сформульовані вимоги до веб-додатку для підтримки процесу технічного обслуговування автомобілів:

1. ПЗ має надавати можливість підтримки процесу технічного обслуговування автомобілів.
2. Можливість оцінювати замовлення в цілому та по окремим пунктам.
3. SPA архітектуру в клієнтській частині.
4. Максимально простий і зручний UI, щоб користувач інтуїтивно розумів подальші кроки.

5. Сторінка профілю користувача, де власник авто може подивитись інформацію про себе, додати та видалити авто, переглянути статус робіт.
6. Модуль підбору СТО поблизу користувача.
7. Модуль для оплати онлайн за допомогою Stripe.
8. Модуль для аналізу характеристик транспортного засобу та підбору сервісів, що будуть корисними.
9. Модуль авторизації та аутентифікації oAuth2.
10. Модуль взаємодії з мапами для відображення місця розташування СТО (Google Maps).
11. Модуль для генерації та розсилки сповіщень на електронну адресу.
12. Модуль API, що зв’язує решту модулів і формує відповіді на запити, що приходять із клієнтської частини.

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ, ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ ВЕБ-ДОДАТКІВ

2.1. Вибір веб-застосунку в якості типу програмного забезпечення

Загалом бізнес-ідею можна реалізувати різними видами програмного забезпечення, наприклад, веб-застосунком, кросплатформенним додатком для Unix чи Windows подібних систем або як мобільний додаток. Перш за все, необхідно визначитися з цільовою аудиторією та потенційними користувачами продукту. Після цього можна починати проєктування, зважаючи на особливості розроблюваного програмного забезпечення, спеціальні вимоги та технічне завдання.

Розглянемо два можливі типи ПЗ: кросплатформенна програма та веб-застосунок.

Веб-застосунки мають низку переваг:

1. Для використання веб-застосунку не потрібні великі обчислювальні потужності та найновіше апаратне забезпечення. Для доступу до такого ресурсу потрібні комп’ютери чи мобільні пристрої із встановленим веб-браузером та підключеним інтернетом.
2. Веб-застосунок частіше оновлюється, але користувачам не потрібно чекати, поки завершиться цей процес, адже всі дії проходять на сервері, а у браузері завантажується нова версія.
3. Незалежність платформи. Не потрібно враховувати усі особливості операційних систем та бібліотек, адже про це подбали розробники браузерів.
4. Можливість впровадження адаптивного чи респонсивного дизайну на екранах із різною роздільною здатністю, не гаючи час на розробку окремих версій для мобільних пристроїв.

5. Швидкість та ціна процесу розробки веб-додатку значно менша, адже деякі мови веб-програмування дозволяють розробляти як клієнтську частину, так і серверну, що дає можливість зменшити кількість розробників.

На жаль, веб-застосунки разом із низкою переваг мають певні недоліки:

1. Необхідність підключення до інтернету з хорошою швидкістю з'єднання.
2. Необхідність регулярного оновлення клієнтського коду, оскільки стандарти міняються досить часто, а життєвий цикл специфічних веб-фреймворків обмежений, як правило, двома роками.
3. Нові технології, що знаходяться у стадії тестування, використані в додатку не мають широкої підтримки у веб-браузерах.

Якщо говорити про програми для настільних комп'ютерів, то варто згадати такі переваги:

1. Можливість оптимізації програмного забезпечення під певну операційну систему.
2. Десктопні програми мають можливість взаємодіяти із структурними елементами комп'ютера, наприклад, із відеокартою чи чи процесором, що дозволяє підвищити швидкість виконання.
3. Інтерфейс операційної системи надає набагато більше можливостей для користувача, ніж інтерфейс веб браузера.
4. Користування десктопною програмою безкоштовне, адже немає потреби оплачувати додаткові сервери, кластери та бази даних.
5. Часто для програм орієнтованих на персональні комп'ютери не критичне дуже повільне підключення до інтернету.

Програмне забезпечення для комп'ютерів також має деякі недоліки:

1. Як правило, програмне забезпечення, встановлене на одному комп'ютері недоступне з іншого.

2. Зазвичай для коректної роботи не достатньо існуючих протоколів, тому необхідно розробляти, тестувати та оновлювати власні, що є досить складним завданням.

Загалом, на сьогодні розробники програмного забезпечення масштабують свої продукти не тільки як веб-застосунки чи програми для персональних комп'ютерів, а й як комплекс програм, орієнтованих на всі найпопулярніші платформи, у тому числі і мобільні пристрої.

Необхідно зазначити, що згадані у підрозділі 1.2 аналоги, створені, існують та підтримуються тільки як веб-застосунки.

Після глибокого та детального аналізу вимог, сформованих у підрозділі 1.3 та огляду поширених типів ПЗ, описаних вище, було обрано реалізацію програмного забезпечення у формі веб-застосунку із клієнт-серверною архітектурою.

2.2. Мови програмування для серверної частини ПЗ

Перед розробкою будь-якого програмного забезпечення необхідно провести огляд існуючих технологій та мов програмування, проаналізувати доречність їх використання у розробці. Серверна (back end) частина веб застосунку складається із коду, що отримує запити від клієнтської частини, опрацьовує їх відповідно алгоритмам, що були описані бізнес логікою, виконує запити до бази даних, реалізує логіку програми.

Найпоширенішими мовами програмування для розробки веб-застосунків на початок 2020 року є JavaScript, PHP, Ruby, Java та Python [2]. Усі ці мови мають широку підтримку розробників, отримують регулярні оновлення та є стабільними. Також вони досить добре себе зарекомендували, адже на даний момент саме на них написана більшість веб-застосунків.

У якості основної серверної мови для розробки back-end частини варто розглянути PHP та Java. Використовуючи фреймворки Spring чи

Hibernate можливо написати веб застосунок мовою Java, а за допомогою Symfony чи Laravel можна створити веб-застосунок мовою PHP.

Відповідно до вимог, поставлених до програмного забезпечення проаналізуємо мови JavaScript та Java оскільки вони є дуже популярними і число розробників, що вивчають дані мови зростає. На кінець 2019 року найуживанішими фреймворками для розробки серверної частини веб-застосунків мовою Java є Spring та Hibernate [3], а для розробки мовою JavaScript – Node.js [4].

Головними пунктами для порівняння цих фреймворків, як серверних є швидкодія та підтримка високої навантаженості: як швидко застосунок відповідає на дію користувача. Порівнюючи швидкодію Java із Node.js, стає зрозуміло, що Node.js є більш оптимальним і швидким рішенням лише при досить значній кількості запитів. Це пояснюється тим, що Node.js працює асинхронно не блокуючи потік вводу даних, а також використовує потужний двигун chrome V8. Саме тому для написання чатів, ботів, веб-додатків, що працюють у реальному часі краще підходить Node.js [5]. У той же час використовуючи Spring ми маємо всі інструменти для розробки повноцінного серверу, на відміну від Node.js, де треба завантажувати додаткові пакети із npm.

Наступним аспектом для порівняння мов Node.js та Java є паралельна робота веб-додатків та масштабованість. Зазвичай під масштабованістю розуміють стабільну роботу веб-додатків зі збільшенням кількості одночасних запитів. Ця властивість є критично важливою для програмного забезпечення, що обслуговує велику кількість користувачів. Node.js є асинхронним та однопоточним, завдяки чому операції вводу та виводу не блокують потік виконання, завершуючись за його межами. Дана властивість гарантує передбачувану масштабованість простих веб-додатків, однак розробки складного ПЗ із безліччю паралельних процесів вимагає глибокого розуміння, значної уваги розробника та досить великого рівня експертизи. Веб-фреймворки написані мовою Java, у свою чергу, працюють у режимі

багатопоточності, що дозволяє використовувати повністю ресурси сервера, а також можливо додати певні пакети, що дадуть змогу обробляти асинхронні дані. Таким чином, мова програмування Java має повний інструментарій для досягнення необхідної масштабованості розроблюваного програмного забезпечення.

Не менш важливим пунктом для порівняння цих мов є обробка помилок. Простота та прозорість обробки помилок є критичним при виборі мови програмування. Як JavaScript, так і Java мають механізми з відловлювання та обробки виняткових ситуацій. Але у більшості випадків оброблення помилок мовою Java простіше та потребує менше зусиль та часу для відлагодження.

Під час вибору мови програмування для імплементації певних програмних рішень необхідно звернути увагу на простоту написання коду та швидкість освоєння нових технологій програмістами. Java є стандартною об’єктно орієнтованою мовою. Розробники мають можливість використовувати патерни ООП, щоб код був легкий для сприйняття та масштабування.

Веб-фреймворк Spring, написаний на Java, має вбудовану підтримку багатьох популярних баз даних, забезпечує надійний захист, легко масштабується та має широку спільноту розробників. Spring також надає можливість написання як монолітного коду, так і мікросервісів, він є зручним для написання досить великих програм, оскільки не нав’язує розробникам певну модель програмування.

Node.js може використовувати велику кількість готових бібліотек із npm, є дуже швидким, не потребує значних обчислювальних потужностей процесора, відмінний для створення Rest API і використовує JavaScript, що є простою мовою програмування. Основними недоліками є відсутність підтримки реляційних баз даних та складність наявних бібліотек для формування запитів, недостатня масштабованість, адже один потік не зможе

обробляти настільки багато даних, як декілька потоків, вкладеність викликів функцій та асинхронність, що часто ускладнює процес відлагодження.

2.3. Технології розроблення front end частини ПЗ

Front end розробка – це процес створення інтерактивних веб-сторінок за допомогою технологій розмітки сторінок HTML, стилізаційного оформлення CSS та скриптів, за допомогою яких відбувається маніпуляція елементами DOM. Основним завданням front end розробки є створення веб-сторінок, що однаково відображаються на різних пристроях, адаптуючись під їх роздільну здатність та підтримуються різними операційними системами. Зважаючи на це, при розробці веб-додатків необхідно звернути особливу увагу на проектування та розробку дизайну.

Найбільш актуальними технологіями для розробки клієнтської частини, що набули широкого поширення є HTML, CSS та JavaScript.

HTML – мова тегів, що використовується для розмітки гіпертексту на сторінках веб-сайтів. Використовуючи HTML розробник може розділити веб-сторінку на певні структурні блоки, а в них помістити певну інформацію. Для інтерпретації у звичний для нас вигляд такі сторінки обробляються браузерами у відповідності до правил, а потім відмальовуються. HTML теги дозволяють додати на сторінку відео, звуки та зображення, саме тому мова розмітки ж такою універсальною та стандартизованою. Документ HTML складається із трьох основних компонентів:

1. Декларація типу – ключове слово на початку документа, яке визначає його тип.
2. Шапка документа – технічні дані та додаткова інформація про документ, призначена для браузера та розміщена між тегами <head>.
3. Тіло документа – основний зміст документа, де знаходяться інші компоненти розмітки.

CSS (Cascading Style Sheets) – технологія стилізації веб-документів, що дає розробнику змогу визначати візуальну складову стандартних елементів сторінки. Наприклад, стилі тексту, прозорість картинки, відступи навколо структурних блоків, положення елементів, позиціонування дочірніх відносно батьківських. CSS надає веб-розробникам контроль над графічною частиною веб сторінки, вбудованими анімаціями та іншими властивостями. За допомогою цієї технології розробники можуть використовувати інкапсуляцію стилів, тобто писати їх у зовнішніх файлах для усього проекту а потім підключати за допомогою посилання у разі необхідності. Також використовуючи CSS змінні можна розробляти веб-застосунки із різними темами оформлення, наприклад, світлою та темною. Можливості CSS зростають завдяки компілюючим обробникам шаблонів LESS та SASS, що надають можливість визначати більш гнучкі змінні, наслідувати класи та більш грунтовно описувати їх властивості, а це в свою чергу значно спрощує процес написання коду розробниками веб-сторінок.

JavaScript є потужним інструментом для розробки front end частини веб-застосунку. JavaScript – це кросплатформна, скриптова, динамічна мова програмування, що використовується для реалізації взаємодії користувача із веб-сторінкою. JavaScript на стороні клієнта надає інтерфейс для маніпуляцій із DOM деревом, додавання та вилучення елементів, модифікування прототипів, дозволяє створювати анімації статичних елементів. Також за допомогою цієї мови програмування можливо обробляти різноманітні форми, слайдери, модальні вікна, реагувати на рухи курсору миші. JavaScript також має вбудовані інструменти для комунікації із серверною частиною.

Існує декілька підходів для реалізації клієнтської частини веб-застосунків: багатосторінкова модель, односторінкова та рендеринг на сервері, коли наступні сторінки приходять користувачу згенерованими із сервера. Оскільки SPA є оптимальним рішенням для розроблюваного

додатку – необхідно вибрати веб-фреймворк, що дасть необхідний інструментарій для реалізації усіх описаних вимог. Найпопулярнішими фреймворками є Angular, React та Vue.

React – веб-фреймворк розроблений компанією Facebook, що дозволяє розробляти односторінкові веб-застосунки будь-якого призначення та масштабу [6]. Перевагами React є: відносна простота, швидкодія, гнучкість, що дає можливість створювати компоненти з нуля, легкий процес оновлення версій, що надають розробники. Проте є і ряд недоліків: JavaScript – як основний інструмент побудови компонентів, не є строго типізованою мовою, що збільшує відсоток помилок при написанні коду, також JSX (суміш JavaScript та HTML) є досить незрозумілим, адже треба контролювати валідність не тільки тегів, а й коду JavaScript.

Angular – веб-фреймоворк, розроблений у Google для побудови односторінкових веб-застосунків [7]. Саме його зазвичай використовують у великих застосунках, де важлива надійність, масштабованість та передбачуваність. Даний фреймворк має такі структурні компоненти:

1. Модулі – узагальнюють певну функціональність, імпортують необхідні компоненти, сторонні модулі, файли роутингу.
2. Компоненти – відповідають за HTML шаблони та CSS DOM-елементів, а так містять логіку їх відображення й поведінки.
3. Директиви – змінюють вигляд елементів DOM, модифікують їх поведінку і не містять в собі HTML шаблонів.
4. Сервіси – реалізують паттерн «Сінглтон», що дозволяє впроваджувати один екземпляр потрібного класа для всіх залежних компонентів.
5. Метадані – необхідні дані, що використовує веб-додаток.

Рис. 2.1 демонструє взаємодію основних модулів Angular фреймворвка та його внутрішню структуру.

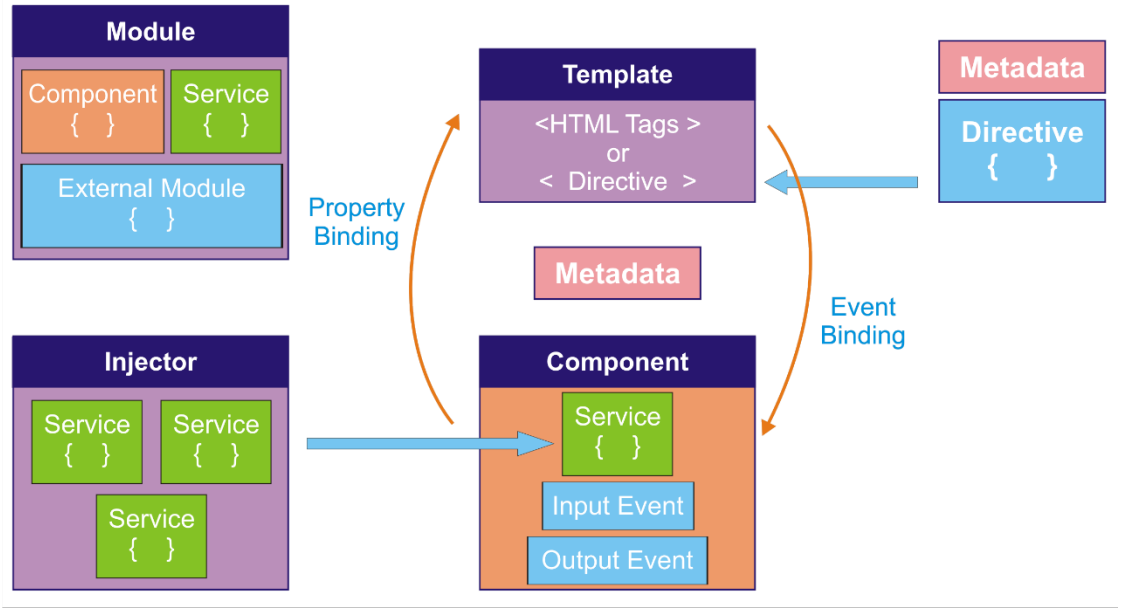


Рис. 2.1. Взаємодія основних модулів фреймворка Angular

Angular має вбудований cli, що допомагає генерувати структурні компоненти програмного забезпечення, також даний фреймворк має строге розділення бізнес-логіки, стилів та власне шаблонів. Для реалізації бізнес-логіки у фреймворк інтегрована мова TypeScript, що унаслідувала усі переваги як нетипізованих так і типізованих мов. TypeScript синтаксично дуже схожий на JavaScript, що дає змогу не витрачати багато часу на опанування. Також, Angular регулярно оновлюється та підтримує оновлення через cli. Для веб-застосунків, де потрібна інтернаціоналізація фреймворк надає вбудовані інструменти, що з легкістю дозволяють виконати переклад сторінок. Завдяки ООП та використанню патернів програмне забезпечення написане за допомогою Angular може бути масштабованим. Однак, варто відмітити декілька недоліків даного фреймворка: велика різноманітність структурних елементів, різноманіття патернів, ООП, необхідність налаштування проєкту та ін.

Vue – дещо менш популярний веб-фреймворк, що використовується для створення складних користувацьких інтерфейсів і по своїй структурі є дуже схожим на Angular, проте є менш потужним [8]. Найголовнішою перевагою Vue серед конкурентів є низький розмір скомпільованих файлів,

а головним недоліком – невелике поширення цього фреймворка, що може сповільнювати розробку за відсутності хорошої документації.

RxJs (reactive extensions for JavaScript) – бібліотека для реактивного програмування, що будується на принципі асинхронних потоків даних [9]. Ключовим поняттям у RxJs є Observable – об'єкти чи функції, що віддають потоки даних з плином часу. Їх особливість в тому, що для отримання даних на них треба підписатися, а для розірвання зв'язку – відписатися. Observer – це об'єкт чи функція, що знає, як обробити послідовності даних. Subscriber – об'єкт чи функція, що пов'язує Observable із Observer. На рис. 2.2 зображено механізм роботи Observable.

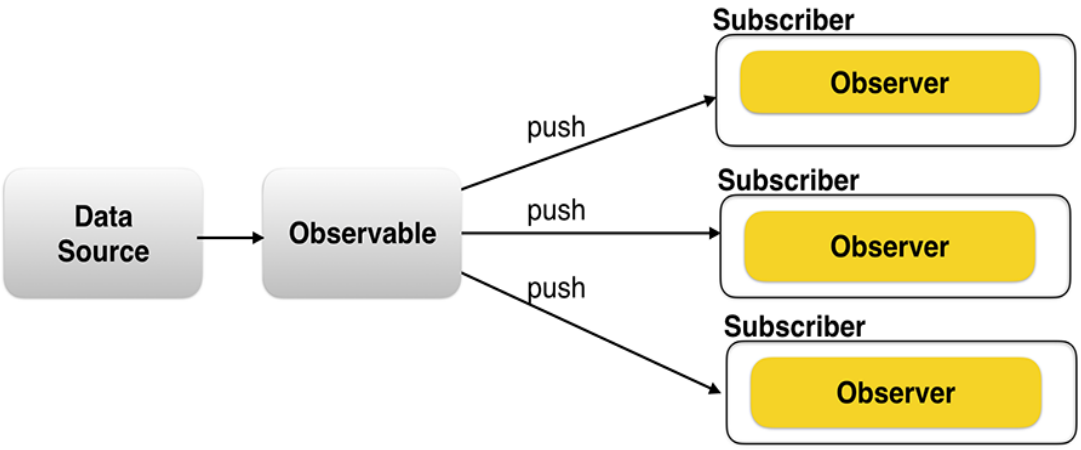


Рис. 2.2. Механізм роботи RxJs

Bootstrap – популярний CSS фреймворк для стилізації інтерфейсу веб-застосунку побудований головним чином на LESS і jQuery [10]. З його допомогою можна створити адаптивний та респонсивний дизайн сторінок веб-додатка, що будуть змінювати розміри елементів динамічно в залежності від пристрою користувача. Більшість сучасних браузерів підтримують Bootstrap, серед них Safari, Chrome, Mozilla. Фреймворк надає розробниками – сітку із 12 стовпців для розташування елементів на сторінці, що спрощує процес розмітки сторінок. Також фреймворк цінний наперед визначеними стандартними стилями оформлення кнопок, форм, заголовків, що є також адаптивними та цілком сучасними в розрізі дизайну. Проте, для

великих комерційних проєктів використовують унікальні CSS класи, які розробляють дизайнери, а від Bootstrap можуть залишити лише сітку.

2.4. Аналіз вибору СУБД

Проаналізувавши початкові вимоги до веб-додатку, було сформовано такі вимоги до СУБД, що використовується серверною частиною:

1. Реляційна модель даних, що дозволить робити складні вибірки даних та чітко побудувати зв’язки між таблицями.
2. Підтримка індексації для прискорення пошуку.
3. Підтримка повнотекстового пошуку.
4. Безкоштовна СУБД з відкритим кодом.
5. Підтримка шардингу та реплікації.
6. Придатність для розгортання у хмарних сервісах.
7. Масштабованість.

MySQL

MySQL – система управління базами даних, що розроблена компанією Oracle і є безкоштовною у розповсюдженні [11]. Написана дана СУБД з використанням мов C++ та C, що робить її сумісною із більшістю Windows та Unix подібних операційних систем.

Основними перевагами MySQL є:

1. Відповідність стандартам ACID.
2. Наявність великої кількості документації та невелика відмінність синтаксису від стандарту SQL.
3. Дана СУБД вміщує практично будь-яку кількість даних: у документації вказано, що MySQL дозволяє зберігати до 50 млн. записів.
4. Висока швидкість виконання запитів.
5. Масштабованість, що проявляється у підтримці роботи з великими базами даних.

6. Система контролю доступу до даних, шифрування при їх передачі.
7. Можливість шардування та реплікації.
8. Клієнт-серверна архітектура: до одного сервера MySQL можуть під'єднатися додатки для отримання даних.

Серед недоліків MySQL зазвичай згадують:

1. Налаштування горизонтального масштабування бази даних є досить складним завданням, що потребує досить багато досвіду
2. Зниження швидкості виконання складних запитів.
3. Складність налаштування деяких вбудованих операцій, таких як резервне копіювання.

Особливу популярність дана СУБД здобула серед розробників, що використовують мову програмування PHP, сервери Apache та Nginx. Також вона є частиною LAMP стеку.

PostgreSQL

PostgreSQL – досить популярна реляційна СУБД із відкритим кодом, що часто використовується при створенні веб-додатків [12]. Вона займає велику частину ринку СУБД поруч із MySQL, адже була однією із перших систем управління базами даних. PostgreSQL так само, як і MySQL є кросплатформною та надає можливість для використання вбудованих інструментів обробки даних.

Серед переваг даної СУБД варто зазначити:

1. Підтримка JSON.
2. Можливість створювати нові типи даних.
3. Відповідність стандарту ACID.
4. Підтримка виконання коду всередині самої СУБД.
5. Інструмент для управління PgAdmin.
6. Наявність транзакцій.

До недоліків варто віднести:

1. Зниження швидкодії системи управління базами даних при великих групових операціях.
2. Неочевидність налаштування і встановлення конфігурації даної СУБД.
3. Нижча швидкість вставки даних у таблиці.

Після проведеного аналізу двох найпопулярніших СУБД очевидно, що кожна з них тим, чи іншим параметром задовольняє сформовані вимоги [13]. Водночас MySQL має простішу структуру, легко адмініструється, досить швидка у вставці даних, що є критично важливим, а також може бути легко налаштована на віддаленому сервері чи у хмарному сховищі. Враховуючи ці переваги саме дана СУБД була обрана для розробки серверної частини веб-додатку.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

3.1. Загальна структура веб-додатку

Програмне забезпечення реалізоване у вигляді веб-додатку. Таку програмну систему можна умовно поділити на дві умовні частини – клієнтську (front end) та серверну (back end). Кожна з них виконує свою роль, має користувачів та відповідно функціональні та нефункціональні вимоги.

Серверна частина розроблюваного веб-додатку виконує такі функції:

1. Обробка асинхронних http запитів від front end частини веб-додатку.
2. Реалізація архітектурного патерну MVC.
3. Робота із базою даних для зчитування, збереження, оновлення та видалення даних.
4. Спілкування з Unity API для обробки інформації про автомобілі.
5. Підбір рекомендованих сервісних робіт базуючись на пробігу авто.
6. Здійснення оплати.
7. Надсилання сповіщень користувачам.
8. Пошук СТО поблизу вказаної адреси.
9. Валідація даних, що приходить із клієнтської частини.
10. Захист інформації від несанкціонованого доступу.

Клієнтська частина виконує наступні функції:

1. Реалізація інтерфейсу для кінцевих користувачів.
2. Реалізація патерну MVVM.
3. Формування асинхронних http запитів для динамічного оновлення даних у додатку.
4. Валдація даних введених користувачами.
5. Захист від найбільш поширених веб-атак.

6. Візуалізація розміщення СТО на карті з можливістю їх вибору.
7. Реєстрація профілю користувача та підтвердження електронної пошти.
8. Реалізація адаптивного консистентного дизайну сторінок застосунка на пристроях із різною роздільною здатністю.

Структурна схема системи (див. Додаток 1) розробленого веб-додатка складається із багатьох елементів, кожен із яких пов’язаний із іншим та має визначену наперед роль.

У серверній частині веб-додатку існують такі модулі:

1. Модуль для валідації даних пошуку – призначений для перевірки інформації, що внесена користувачем – поштового індексу та номерного знака автомобіля.
2. Модуль авторизації – створений для реєстрації та аутентифікації користувачів. Даний модуль обробляє дані користувачів, входять у свої облікові записи всередині додатку і створює сесію на сервері, щоб далі відправити необхідні дані клієнтській частині.
3. Модуль підбору продуктів – робить запити у базу даних та модуль, що працює з Unity API для пошуку релевантних продуктів, що можуть бути необхідні даному автомобілю при певному значенні пробігу.
4. Модуль геолокації – виконує пошук найближчих до клієнта СТО та віддає інформацію, про адресу, координати, години роботи сервісного центру та наявні часові періоди для здійснення сервісних робіт.
5. Модуль оплати – дозволяє здійснювати безпечну оплату вибраних продуктів та сервісів за допомогою банківських карток. Також даний модуль обробляє можливі знижки та додає їх до кошика користувача під час бронювання, а на етапі оплати знімає суму з урахуванням спеціальних пропозицій.

6. Модуль зв’язку з Unity API – виконує запити у систему Unity для отримання списку автомобілів та водіїв, а також актуальні продукти, що доступні у сервісних центрах.
7. Поштовий модуль – здійснює відправку сповіщень на пошту користувачам. Також даний модуль надсилає повідомлення про зміну пароля та різноманітні акції та знижки.
8. Модуль інтеграції з Uber – надає інформацію про водіїв, що авторизуються через систему Uber, їх автомобілі та спеціальні бонуси від додатку.
9. Модуль формування звітів – надає можливість завантажити звіти по виконаним роботам.

У клієнтській частині розроблено такі модулі:

1. Модуль підбору продуктів та сервісів – дозволяє користувачу переглянути різні категорії робіт та доступних сервісів, щоб обрати потрібні.
2. Модуль вибору СТО – відображає дані про станції технічного обслуговування та дилерські центри, наявність вільних місць та дозволяє переглядати дані як у вигляді списку, так і як маркери на мапі.
3. Модуль бронювання – дозволяє клієнту здійснити бронювання обраних сервісів.
4. Модуль кошика – відображає інформацію про бронювання, вартість, додаткові коментарі та промокоди.
5. Модуль оплати – надає можливість вводу банківської картки та оплати обраних сервісних робіт онлайн.
6. Модуль рейтингів – дозволяє залишати оцінки та коментарі на виконані замовлення.
7. Модуль управління автомобілями – дозволяє знайти автомобіль по номерному знаку, додати новий автомобіль та видалити старий.

Розроблений програмний застосунок передбачає інтеграцію з компаніями, що володіють великими автопарками, такими як Uber, Bolt та іншими. За допомогою профілів цих компаній водії зможуть реєструватися у системі та бачити свої дані, що будуть передаватися компанією партнером.

Дана інтеграція дозволить:

1. Збільшити кількість користувачів системи.
2. Полегшити авторизацію через системи компаній партнерів.
3. Збільшити потік автомобілів на СТО.
4. Отримувати від СТО знижки, які будуть надаватися водіям у вигляді системи лояльності.
5. Витрачати менше часу на планування регламентних робіт та перевірку сервісних центрів.

3.2. Структура бази даних веб-додатку

Для розроблюваного програмного забезпечення було прийняте рішення про використання реляційної бази даних, оскільки вона є надійною, дозволяє робити складні вибірки даних та є має можливість масштабування й шардування. Структура бази даних розроблюваного веб-додатку зображена у вигляді діаграми сутність-зв’язок (див. Додаток 2), що описує типи зв’язків між таблицями, первинні та зовнішні ключі, типи полів таблиць. Усі типи полів, зазначені на діаграмі, є визначеними у реляційній СКБД MySQL.

У створеній базі даних визначені такі таблиці: Customer, Personal Profile, Job, Basket, Garage Appointment, Payment Method, Payment Status, Unity Vehicle Details, Garage, Appointment Slot, Product, Price. Усі таблиці приведені у першу, другу та третю нормальну форму, тобто у кожній таблиці видалено поля, що не залежать від жодного ключа або є однаковими. Таблиці, що відносяться до декількох записів були пов’язані зовнішніми ключами.

Таблиця Customer зберігає дані про користувача як клієнта системи. Також вона посилається на таблицю Personal Profile обов’язковим зв’язком один-до-одного для ідентифікації у якого користувача які персональні дані. У цій таблиці зберігаються такі дані:

1. Чи активований профіль – поле, яке містить булеве значення і змінюється після того, як користувач перейшов за посиланням у листі підтвердження.
2. Персональний профіль користувача – дані про водія: ім’я та прізвище, електронну пошту, номер мобільного дозвіл на сповіщення через СМС та електронну пошту.

Таблиця Personal Profile зберігає особисті дані кожного користувача:

1. Ідентифікатор користувача.
2. Згоду користувача із правилами користування та політикою конфіденційності – булеве значення, що власник авто може змінити в налаштуваннях.
3. Дозвіл на отримання сповіщень на електронну адресу – булеве значення, що власник авто може змінити в налаштуваннях.
4. Дозвіл на отримання сповіщень на мобільний пристрій – булеве значення, що власник авто може змінити в налаштуваннях.
5. Ім’я та прізвище – текстові поля.
6. Номер мобільного – текстове поле.

Таблиця Job має повну інформацію про замовлення, що здійснив користувач через веб-застосунок. Посилається на таблиці Personal Profile та Payment Method обов’язковим зв’язком багато-до-одного для ідентифікації замовлень та методів оплати користувача. Також посилається на таблиці Basket, Garage Appointment, Payment Status та Vehicle Details обов'язковими зв’язками один-до-одного. Містить таку інформацію:

1. Вид акаунту користувача – звичайний чи під'єднаний до Uber.
2. Кошик із продуктами – сервіси, що обрав користувач.
3. Ідентифікатор замовлення.

4. Коментар для агентів служби підтримки – необов’язкове текстове поле, що може бути пустим.
5. Ідентифікатор користувача.
6. Оцінку замовлення від користувача – цифрове поле зі значенням від 1 до 5.
7. Пробіг авто користувача – цифрове поле з діапазоном 1 - 1000000.
8. Широта та довгота – текстові поля, що містять координати СТО.
9. Поштовий індекс – текстове поле.
10. Категорія сервісних робіт – текстове поле, що містить один із ключів – MOT, Servicing, Repairs чи Diagnostics.
11. Коментар для працівників СТО підтримки – необов’язкове текстове поле, що може бути пустим.
12. Статус замовлення – текстове поле, що містить один із ключів – draft, abandoned, booked, in progress, completed, cancelled.
13. Посилання на бронювання в системі Unity – текстове поле, унікальний ідентифікатор.
14. Посилання на автомобіль – текстове поле, ідентифікатор автомобіля.
15. Ваучер на знижку – необов'язкове текстове поле.

Таблиця Basket посилається на таблицю Product обов'язковим зв’язком один-до-багатьох і містить інформацію про сервіси, що обрав користувач та вартість замовлення:

1. Ідентифікатори замовлених сервісів – унікальні об’єкти текстового типу, що відповідають обраним товарам.
2. Ціна – цифрове поле в діапазоні 0 - 10000.

Таблиця Garage Appointment має посилання обов’язковим зв’язком один-до-одного на таблиці Garage та Appointment Slot. Містить дані про замовлення:

1. Дата сервісних робіт – поле типу дата, що містить дані про обраний час проведення робіт.

2. Ідентифікатор СТО, де будуть проходити роботи.
3. Інтервал дня – текстове поле що містить один із двох варіантів «перша» чи «друга».

Таблиця Payment Method зберігає дані банківських карт користувачів:

1. Тип банківської карти – текстове поле, що містить бренд компанії виробника картки (Visa/Mastercard).
2. Країна – текстове поле, що містить назву країни, де проходить ТО.
3. Валюта – текстове поле, містить скорочену назву валюти, що використовується для сплати.
4. Строк дії картки (місяць та рік) – текстове поле.
5. Чотири останні цифри номеру картки – текстове поле.

Таблиця Payment Status зберігає інформацію про платежі:

1. Сума для списання – цифрове поле з діапазоном 0 - 10000.
2. Сума, яку вдалося списати – цифрове поле з діапазоном 0 - 10000.
3. Сума, що була отримана – цифрове поле з діапазоном 0 - 10000.
4. Сума, до підтвердження – цифрове поле з діапазоном 0 - 10000.
5. Сума, що повинна бути отримана – цифрове поле з діапазоном 0 - 10000.
6. Статус платежу – текстове поле, що містить один із стандартних ключів, що повертає Stripe API.

Таблиця Unity Vehicle Details містить дані про автомобілі:

1. Ідентифікатор – унікальне текстове поле, за допомогою якого можна отримати дані про автомобіль із Unity.
2. Марка – текстове поле, назва виробника транспортного засобу.
3. Модель – текстове поле, модифікація транспортного засобу.
4. Номерний знак – унікальне текстове поле, що містить напис із номерної таблички авто.
5. Колір – текстове поле, містить назву кольору авто.

Таблиця Garage містить дані про СТО:

1. Адреса – текстове поле, що містить адресу СТО, яку надає Unity.
2. Назва – текстове поле, що містить назву СТО, яку надає Unity.
3. Години роботи – текстове поле, що зберігає дані про час роботи СТО, які надає Unity.

Таблиця Appointment Slot зберігає дані про час бронювання:

1. Назва слоту – текстове поле, що може бути лише «ранок» чи «день».
2. Час початку робіт – цифрове поле, що містить час початку робіт.
3. Час завершення – цифрове поле, що містить час завершення робіт.
4. Опис робіт – текстове поле, що містить назви робіт, які замовив користувач.

Таблиця Product посилається обов'язковим зв'язком один-до-одного на таблицю Price та містить інформацію про наявні сервіси на СТО доступні користувачу:

1. Повний опис – текстове поле, що надає Unity, яке містить повний опис робіт, включених в обраний продукт.
2. Рекомендований пробіг автомобіля – цифрове поле, яке надає Unity, що містить пробіг авто, рекомендований для даних робіт.
3. Ціна – цифрове поле, що містить ціну обраного продукту.
4. Категорія сервісних робіт – текстове поле, що містить один із ключів – MOT, Servicing, Repairs чи Diagnostics.
5. Короткий опис – текстове поле, що надає Unity, яке містить короткий опис робіт, включених в обраний продукт.

Таблиця Price містить дані про ціну сервісних робіт:

1. Сума знижки – цифрове поле, що містить фіксовану знижку на певний продукт.
2. Ціна без ПДВ – цифрове поле, що містить вартість замовлення без урахування ПДВ.

3. Ціна з ПДВ – цифрове поле, що містить вартість замовлення з урахуванням ПДВ.
4. ПДВ – цифрове поле, що містить ПДВ.

3.3. Модуль підбору рекомендованих сервісних робіт

Розроблюваний веб-додаток передбачає створення модулю підбору сервісних робіт та продуктів для конкретного авто. Модуль підбору рекомендованих сервісних робіт веб-додатку використовує модуль бази даних, а також модуль інтеграції з Unity. При виборі відповідно сервісних робіт, ТО чи діагностики на клієнтській частині – сервер отримує запит, що містить дані про авто і базуючись на моделі, пробігу та даті останніх сервісних робіт, виконує запит у базу даних та на сервер Unity для отримання списку релевантних продуктів. Користувач має можливість оновити пробіг автомобіля та зробити запит на оновлення рекомендованих продуктів. У цей час інтерфейс перемальовується, оскільки сервер повернув нові продукти. Якщо користувач хоче переглянути усі наявні види сервісних робіт для свого автомобіля, достатньо натиснути кнопку «Show more». При цьому у список сервісів, що вже відображаються, додається два продукти – для меншого значення пробігу та для більшого. Після того, як власник транспортного засобу обрав необхідні сервісні послуги і здійснив замовлення – база даних оновлюється актуальною інформацією.

3.4. Модуль генерації звітів

Розроблюваний веб-додаток також передбачає створення модулю для формування звітів із інформацією про замовлення, виконані роботи та ін.

Модуль формування звітів використовує модуль бази даних. Усі звіти формуються на замовлення зі статусом «completed», оскільки це реальні файли, які користувач може завантажити на будь-який свій пристрій і переглянути у разі необхідності. Дані файли створюються модулем генерації через добу після зміни статусу замовлення. Модуль формування

звітів здійснює запит до модуля інтеграції Unity API, отримує актуальні дані щодо замовлення і оновлює інформацію в базі даних веб-додатка. Сформовані звіти містяться у веб-додатку у вигляді pdf файлів, які зберігаються в S3 bucket сховищі. Після того, як звіт стає доступним, у клієнтській частині спрацьовує директива *ngIf з async pipe і стає активним посилання для завантаження або друку звіту, що містить таку інформацію:

1. Дані про автомобіль та власника.
2. Список сервісних робіт, що були виконані.
3. Ціна виконаних робіт.
4. Адреса СТО, контактний номер, дата та час сервісного обслуговування.
5. Коментарі замовника та майстрів СТО.

Веб-додаток також надає можливість переглянути короткий звіт, перейшовши на сторінку замовлень та обравши те, що цікавить користувача.

3.5. Модуль інтеграції з Unity API

Основним джерелом даних, що використовуються в додатку є систем Unity. Це CRM система, де агенти вносять дані про автомобілі, їх власників, пройдені ТО, та інші. Для того, щоб СТО з’явилася в даній системі, необхідно надати всі документи, що підтверджують легальність, а також контракти із офіційними дилерами запасних частин. Таким чином, користувачі можуть бути впевнені, що усі майстерні є верифікованими, легальними та атестованими.

Даний модуль інтеграції є центральним вузлом між Unity та розроблюваним веб-додатком. При кожному запиті до CRM вона бере дані із своєї власної бази та здійснивши необхідну фільтрацію чи попередню обробку, повертає їх модулю взаємодії з Unity API. Схема взаємодії зображена на рис. 3.1.

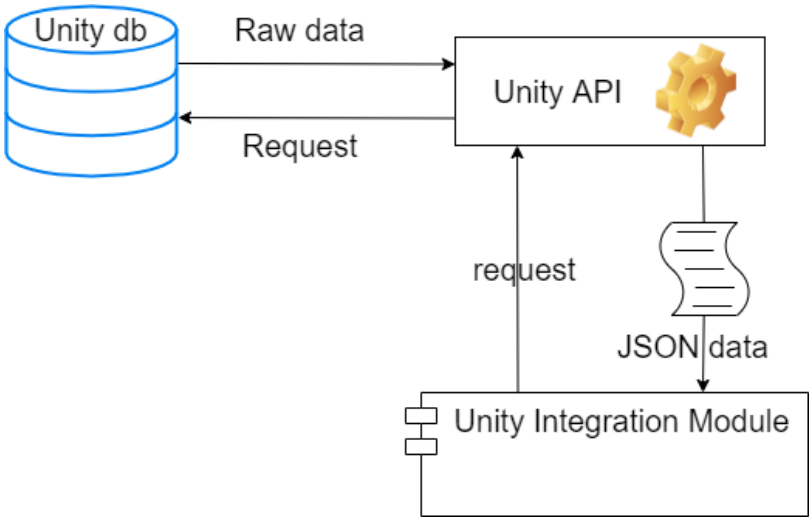


Рис. 3.1. Зв’язок модулю інтеграції з Unity API та системи Unity

Як і інші, модуль інтеграції використовує модуль бази даних для збереження та оновлення інформації про автомобілі. Відповідно списки усіх послуг, наявних на СТО, міститься також у Unity і модуль інтеграції з Unity API виконує запити для того, щоб отримати необхідні дані.

Unity CRM є також порталом для агентів підтримки, які можуть спілкуватися із користувачами, обробляти їх заявки на проходження ТО, додавати знижки, коригувати дані користувачів, тощо.

3.6. REST API модуль

Серверна та клієнтська частини обмінюються даними використовуючи методи GET, POST, PATCH, DELETE. На сервері відповідно існує модуль, що відповідає за точки входу API та визначає унікальні шляхи, на які клієнтська частина надсилає відповідні запити. Рис. 3.2 ілюструє схему роботи REST API.

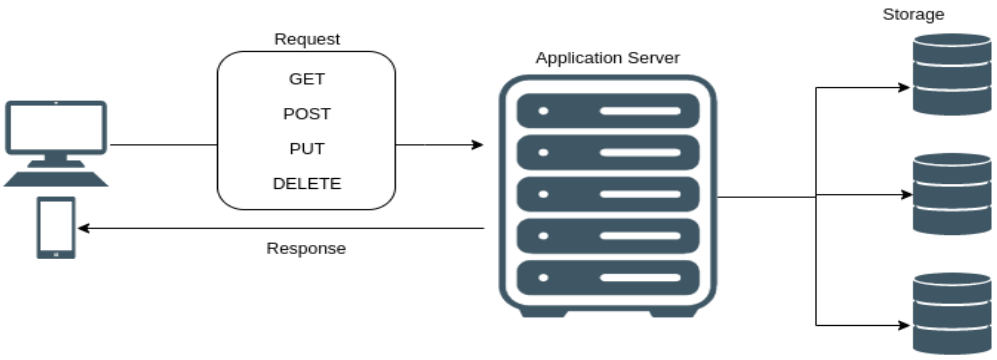


Рис 3.2. Схема взаємодії REST API

Оскільки кожна операція захищена від несанкціонованого доступу – у запиті мають обов'язково бути наявні заголовки із ключем (токеном), що перевіряється модулем аутентифікації. Відповідь сервер надсилає у форматі JSON. Клієнтська частина в свою чергу має вбудований HttpClientModule, що містить всі необхідні методи та інтерфейси. В результаті здійснення запиту на сервер клієнт отримує Observable із даними, на які можна підписатися та отримати їх у чистому вигляді. Дана абстракція є досить ефективною, адже навіть якщо було здійснене помилкове звернення на сервер, то допоки у Observable нема підписників – дані не віддаються. Усі Http запити винесені у окремі сервіси для повторного використання та дотримання принципу DRY [14]. Усі сервіси у веб-додатках написаних на Angular працюють за принципом впровадження залежностей – усі залежності реєструються у масиві providers потрібного модуля і при ініціалізації додатку кожен компонент та сервіс, що використовує залежності – отримує один екземпляр залежностей. Даний підхід запроваджений для зменшення використання пам'яті та використання патерну «Singleton».

3.7. Механізм оптимізації відображення DOM дерева

При розробці веб-додатків важливою складовою є плавність зміни станів DOM дерева, адже незрозуміла поведінка інтерфейсу може змусити користувача покинути сторінку. У розроблюваному додатку використовуються всі три види директив, наявні в Angular – структурні,

атрибутні та, власне, компоненти. Компоненти – це шаблони сторінок, що містять бізнес-логіку, стилізацію та HTML розмітку. Всередині компонентів дуже часто використовуються структурні директиви ngFor, що дозволяє ітерувати по масиву об’єктів, додаючи їх відображення у DOM та ngIf, яка дає можливість додавати й виключати елементи. За замовчанням дані директиви є синхронними, тому вони не можуть працювати напряму із відповідями сервера, які є Observable, тому було прийняте рішення використати вбудований механізм async pipe для надання асинхронності. Також цей механізм допомагає оновлювати навігаційну частину додатку та відображати дані про користувача, після того, як він увійшов у свій аккаунт. Щодо атрибутних директив – вони використовуються менше, наприклад, для того, щоб округлити значення ціни та відображати десяті частини дещо вище цілих.

3.8. Механізм маніпуляції та збереження даних клієнтською частиною

Для збереження стану клієнтської частини веб-додатку використовується Redux архітектура [15]. Існує ціла низка менеджерів стану, що надають гнучкі можливості налаштування, проте, у розроблюваному програмному забезпеченні ключовим фактором є розмір кінцевого файлу, тому було прийняте рішення зробити власну реалізацію сховища, схожу на деякі аналоги, але з використанням лише тих частин, що дійсно потрібні. Для того, щоб збільшити швидкодію додатка, при реалізації менеджеру стану веб-додатка було використано RxJs, а саме: Observabe, BehaviorSubject, Subject та чимала кількість операторів. Загалом стан веб-додатка у конкретний момент можна описати у вигляді масиву із полями, які змінюються під час вибору продуктів, часу проходження ТО та інших параметрів. Були створені такі структурні компоненти: редюсери для оновлення стану, селектори для отримання даних із сховища та дії – для виклику редюсерів після якихось змін стану. Таким чином, у будь-якому

компоненті можна дістати актуальні дані з глобального сховища, та відобразити їх користувачу, а також асинхронно оновлювати їх, при зміні користувачем параметрів бронювання рис. 3.3 ілюструє принцип роботи розробленого сховища.

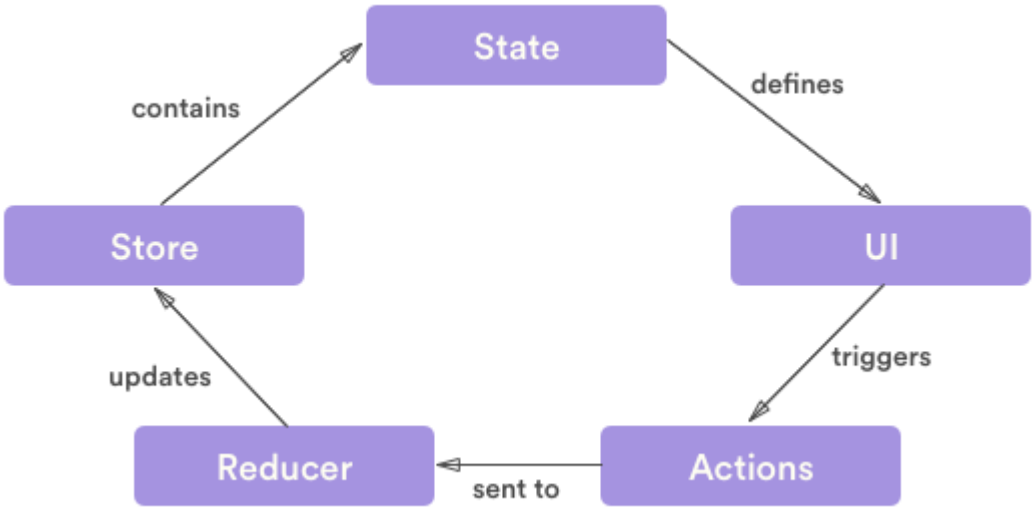


Рис. 3.3. Механізм роботи менеджеру стану веб-додатку

3.9. Завантаження даних перед ініціалізацією компонентів та веб-додатку в цілому

Оскільки розроблюваний веб-додаток має вбудовані мапи google, інтеграцію із сервісом для оплати Stripe [16], аналітику – необхідно зберігати токени для авторизації у інтегрованих сервісах. Для цього перед ініціалізацією веб-додатка необхідно зробити запит на сервер і завантажити звідти параметри, оскільки на клієнтській частині зберігати їх небезпечно. Даний механізм був реалізований за допомогою вбудованого в Angular ініціалізатора, який викликає функцію, що у свою чергу робить запит на сервер і зберігає відповідь у глобальному сховищі. Також для початкового відображення деяких компонентів, наприклад з рейтингами чи зі списком наявних СТО, необхідна певна інформація, яку потрібно отримати від сервера. Для цього у Angular існують резолвери, які виконують запит на отримання даних, і не відмальовують компонент, поки не отримана

відповідь сервера. Оскільки така поведінка не є зрозумілою користувачу – було прийнято рішення при кожному запиті показувати анімацію завантаження у вигляді плоских кульок, що рухаються по колу. Таким чином був забезпечений необхідну функціональність і покращений UX веб-додатку.

3.10. Генерація сервісів для клієнтської частини

Для того, зробити запит на сервер – клієнтська частина має вказати точку входу та усі необхідні параметри та ключі. Очевидно, що під час написання модулів серверної частини деякі точки входу та параметри обробників змінюються, саме це найчастіше є причиною того, що дані не відображаються або і взагалі не приходять на клієнту. Для уникнення даних проблем було прийнято рішення впровадити інструмент, що автоматично генерує методи для сервісів Angular, базуючись на специфікації, описаній розробником модулю на сервері. Таким чином, у згенерованих методах будуть враховані усі особливості і при розробці клієнтської частини буде виникати помилка на етапі компіляції, що буде вказувати на необхідність оновлення сервісів. У розроблюваному програмному забезпеченні дану функціональність було реалізовано засобами Swagger та Maven code generator [17]. Для цього згідно з документацією swagger був створений файл конфігурацій та опису API, який реагує на внесення поправок у існуючий код. Крім цього даний інструмент дозволяє полегшити тестування, адже надає графічний інтерфейс розробникам, де вказані всі точки входу, необхідні параметри та будь-яка додаткова інформація. Maven у свою чергу дозволяє генерувати сервіси для Angular використовуючи вбудовані плагіни і отримувати сервіси, що є максимально надійними, адже у них виконується чимала кількість перевірок вхідних параметрів на відповідність типів та значень. Насамкінець варто також зазначити, що автоматична генерація коду зменшує кількість помилок, дозволяє розробникам сфокусуватися на бізнес-логіці та автоматизувати процес

оновлення сервісів за допомогою однієї команди, виконаної у середовищі терміналу.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вміст та дизайн сторінок веб-додатку

Для розробки дизайну та прототипування веб-додатку було використано Figma. Ця програма дає можливість створити базові елементи: верхній, нижній та центральний блоки. Для більшої консистентності було створено кожен елемент та зазначено всі параметри, необхідні для відтворення у html та scss. Стилі для кнопок, заголовків, параграфів були винесені у окремий блок зі змінними. Також кожен компонент був перемальований під екрани із різними роздільними здатностями, щоб зробити додаток адаптивним. Загалом структура стилів у додатку є такою:

1. Глобальні стилі для всього додатку, що містяться у корені проєкту.
2. Змінні для кольорів, розмірів, шрифтів.
3. Стилі для кнопок, форм, випадаючих списків, календарів, сповіщень, логотипів.
4. Файл з іконками, що інтегровані в шрифт.
5. Стилі глобальних елементів, таких як хедер, футер, контент, спливаючі вікна, модальні вікна.
6. Стилі окремих компонентів.

Окрему увагу слід звернути на інкапсуляцію стилів, адже за замовчуванням у angular використовується емульована інкапсуляція, що додає кожному DOM елементу унікальний псевдо-клас, за яким до них застосовуються відповідні правила. Оскільки всі класи винесені у окремі файли, то в файлах компонентів майже відсутні правила, що робить легшим подальшу зміну кольорів, розміщення та поведінки елементів. Для відображення іконок можливо використовувати формат svg, але він є неоптимальним, тому в розробленому веб-додатку усі спеціальні символи були об’єднані у шрифт, який можна вставляти в потрібні місця в html документі для відображення.

Загалом у розробленому додатку є такі сторінки:

1. Головна сторінка.
2. Сторінка авторизації.
3. Сторінка реєстрації.
4. Сторінка вибору категорії сервісних робіт.
5. Сторінка сервісів.
6. Сторінка ТО.
7. Сторінка позапланового ремонту.
8. Сторінка діагностики.
9. Сторінка вибору продуктів.
10. Сторінка вибору СТО у списку.
11. Сторінка вибору СТО на мапі.
12. Сторінка додавання банківської картки.
13. Сторінка вибору банківської картки.
14. Сторінка із підсумком бронювання.
15. Сторінка успішного бронювання.
16. Сторінка бронювань.
17. Сторінка окремого бронювання.
18. Сторінка оцінки бронювання.
19. Сторінка із профілем користувача.
20. Сторінка зі списком автомобілів.
21. Сторінка частих запитань.
22. Сторінка допомоги.
23. Сторінка з умовами користування.

Для навігації між сторінками більшою мірою використовуються блоки навігації – нижній (footer) та верхній (header). Вони містять гіперпосилання та дозволяють переглядати сторінки бронювань, профілю, та списку авто.

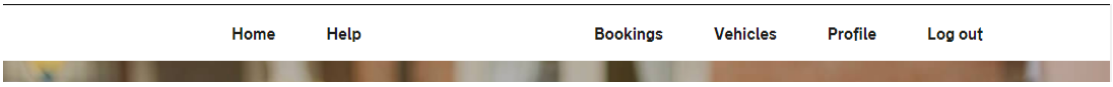


Рис. 4.1. Універсальна навігаційна частина (header)

Рис. 4.2. Універсальна навігаційна частина (footer)

Головна сторінка містить форму, що дозволяє розпочати бронювання після вводу реєстраційного номера автомобіля та поштового індекса. Також на домашній сторінці є опис переваг, які отримує користувач від використання даного веб-додатку.

Сторінка авторизації містить форму для вводу логіна та пароля, яка валідується перед відправкою даних на сервер. Якщо ж користувач ввів неправильну пару логін пароль – під формою з’явиться повідомлення про помилку, а також посилання для скидання паролю, зображене на рис. 4.3.

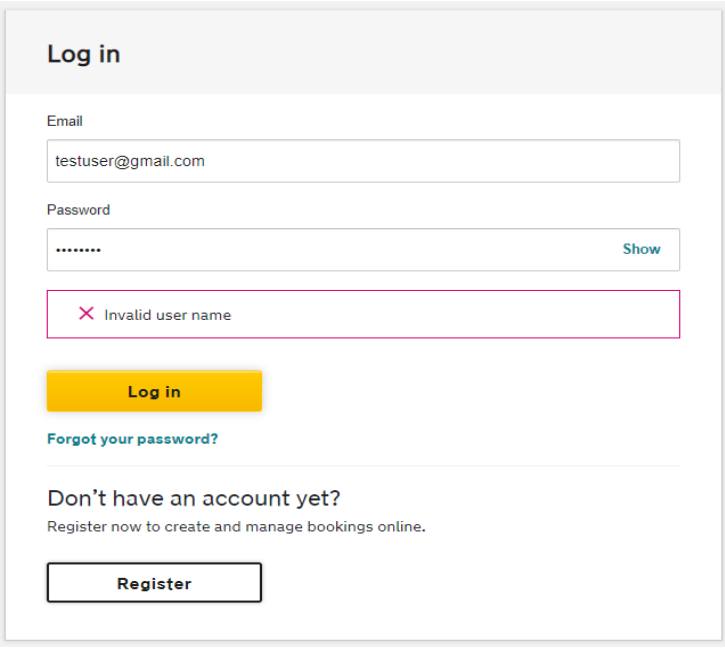


Рис. 4.3. Сторінка авторизації в системі

Після авторизації, користувач має доступ до всіх наявних сторінок. Також з’являється можливість здійснити бронювання без перенаправлення на сторінку авторизації та додати чи видалити автомобіль. Крім того, користувач може перейти на сторінку бронювань, щоб перевірити їх статус та коментарі.

Щоб додати автомобіль, користувач має перейти на сторінку Vehicles та натиснути на кнопку «Add vehicle». Після цього ввести у спливаючому дані із номерного знаку авто та натиснути «Look up» (рис. 4.4).

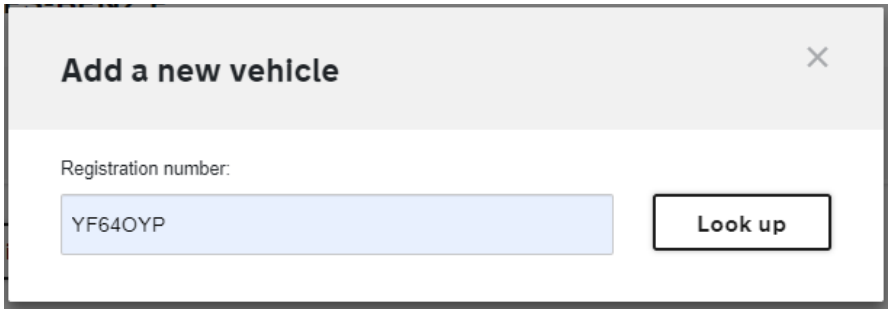


Рис. 4.4. Вікно пошуку автомобіля за номерним знаком

Після того, як автомобіль знайдено – з'являється інформація про нього та кнопка, щоб додати його у свій профіль (рис. 4.5).

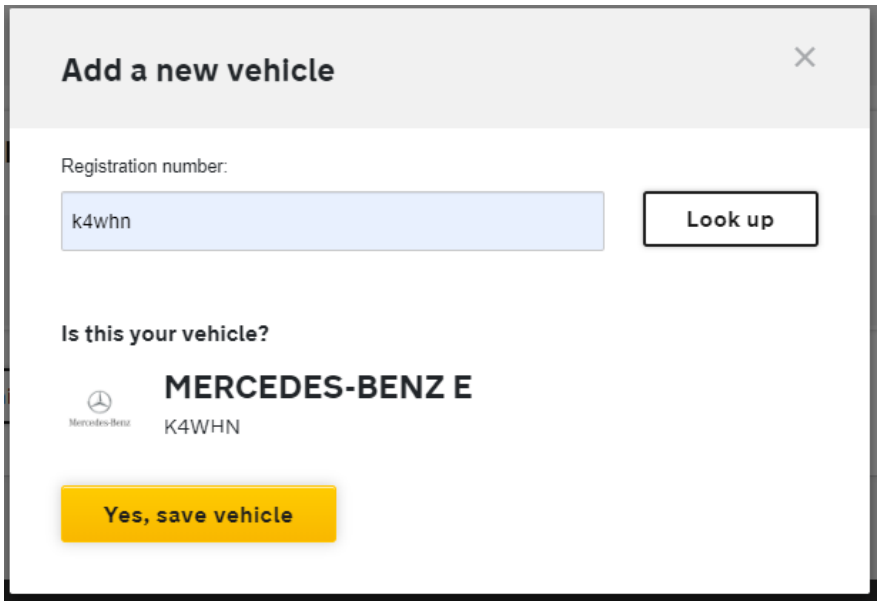


Рис. 4.5. Вікно зв’язування автомобіля з профілем користувача

Після того, як користувач додав більше одного авто – у формі бронювання при введенні номера авто зявлятиметься випадаючий список з усіма наявними транспортними засобами.

Для бронювання ТО користувач має перейти на головну сторінку та ввести дані про авто і поштовий індекс.

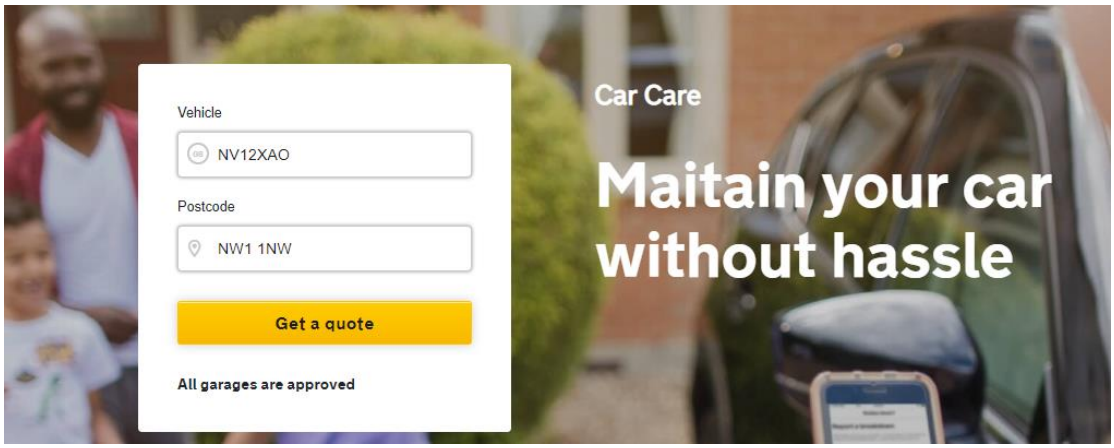


Рис. 4.6. Сторінка початку бронювань

Якщо дані, надані користувачем, правильні – здійснюється навігація на сторінку вибору категорії робіт, зображену на рис. 4.7.

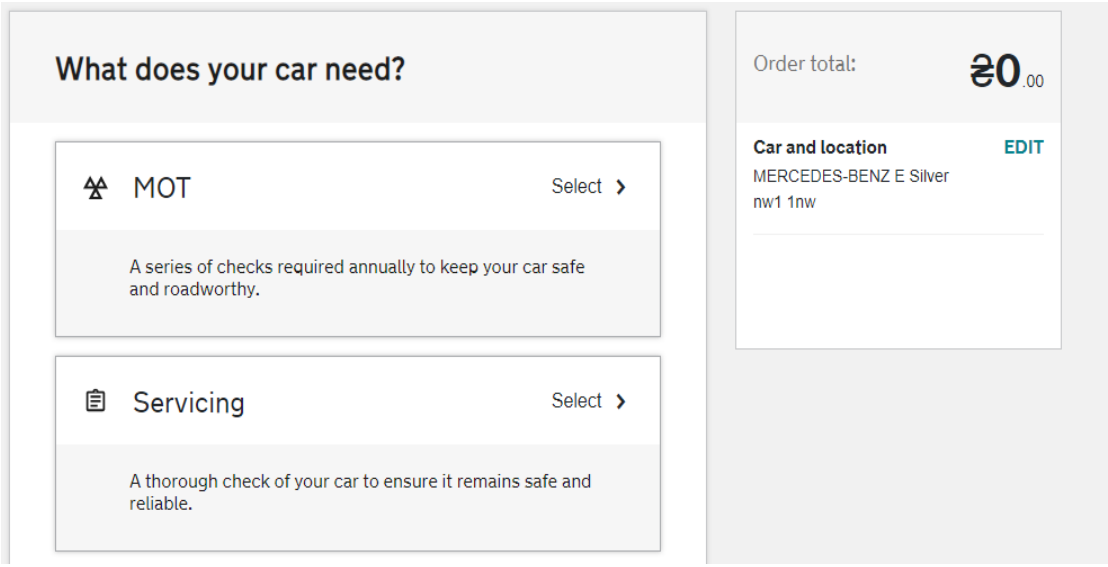


Рис. 4.7. Сторінка вибору категорії сервісних робіт

Як тільки користувач вибрав потрібний пункт, необхідно обрати конкретну послугу, після чого відбувається перенаправлення на сторінку вибору СТО та дати візиту. Усі вільні позиції підсвічуються для більшої наглядності (рис. 4.8).

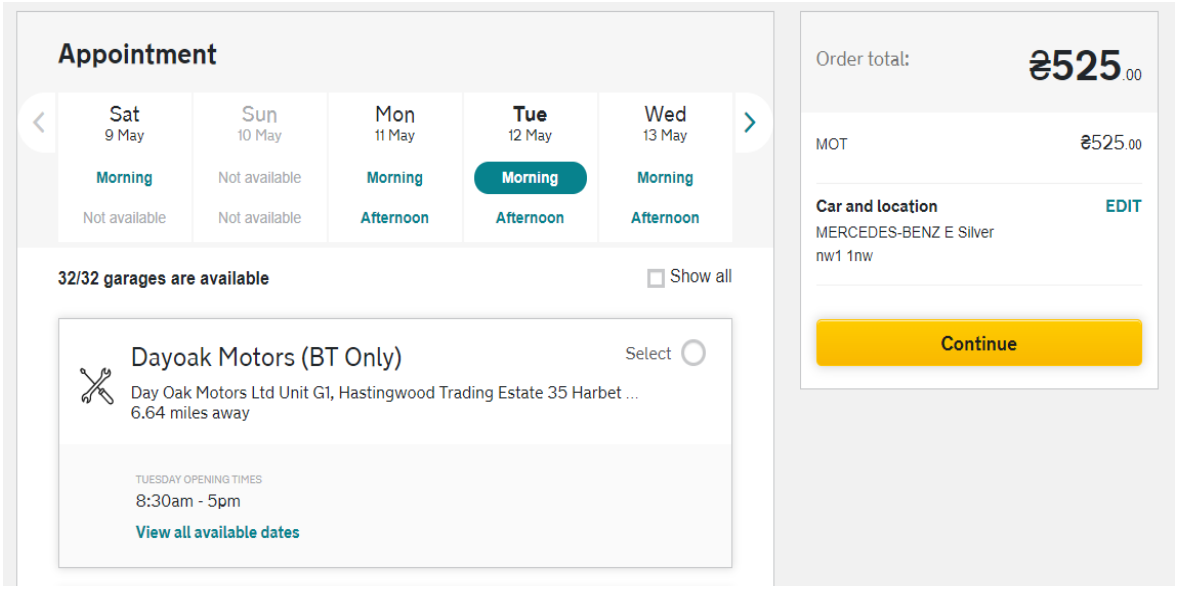


Рис. 4.8. Сторінка вибору СТО для проведення сервісних робіт

Далі відбувається перехід на сторінку оплати, а далі – на сторінку із загальною інформацією про бронювання, де вказані дані водія, його автомобіля, дата та час, коли необхідно прибути для проведення ТО, а також вся інформація про сервіси, що були обрані та їх ціну. Ця інформація дублюється на сторінці My bookings (рис. 4.9).

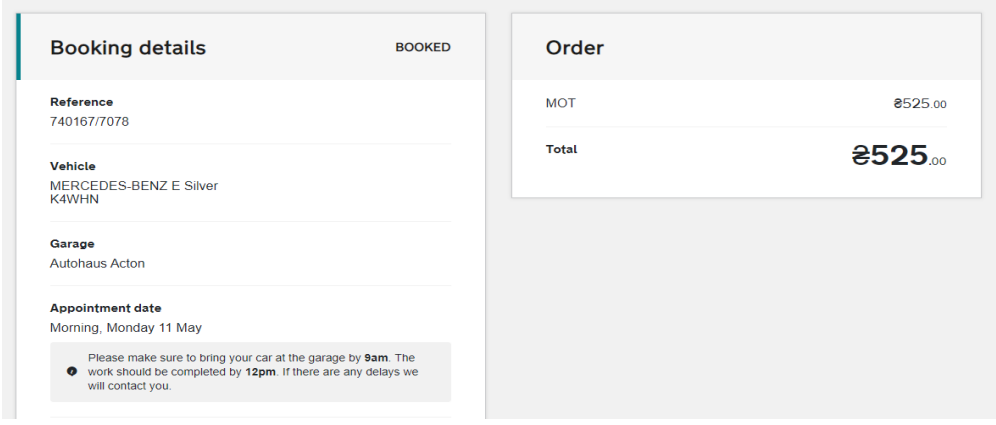


Рис. 4.9. Сторінка перегляду деталей бронювання

4.2. Розробка UX

Ще одним аспектом, що був глибоко досліджений під час розробки, є досвід користування. UX має бути досконалим та інтуїтивним, не викликати додаткових запитань чи обдумування, адже зрозумілість інтерфейсу є ключовим чинником, що може змусити користувача покинути сторінку [18]. Коли користувач заходить на головну сторінку, існує два варіанти – ввести номер авто та поштовий індекс для початку бронювання або увійти в систему. Для першого сценарію існує форма у верхній частині сторінки для пристроїв із великою роздільною здатністю або форма адаптована на весь екран для мобільних пристроїв. У другому сценарії користувач читає текст головної сторінки про переваги рухаючись донизу та бачить в кінці кнопку «Розпочати», яка веде на сторінку авторизації. Таким чином користувача нічого не відволікає, і він переходить далі.

Також важливо забезпечити зрозумілі переходи між сторінками, для цього в додатку є анімація завантаження, що дає змогу користувачу зрозуміти, що треба зачекати. Усі елементи, на які можна натиснути є плавними та анімованими, щоб не збивати з пантелику водія і не змушувати шукати очима, що ж саме змінилося. Верхня частина додатку у версії для комп'ютерів містить назви кроків, які підсвічуються відповідно до етапу бронювання, на якому перебуває користувач.

При виборі сервісів у кожного варіанта є свій значок, наприклад, у діагностики – авто на підйомнику, що є інтуїтивно зрозумілим (рис. 4.10).

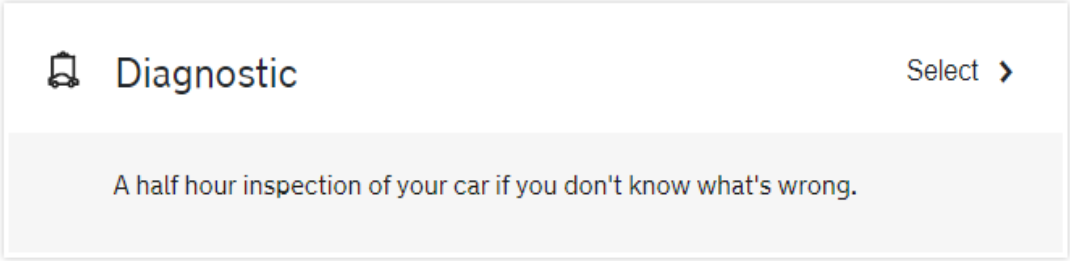


Рис. 4.10. Позначення діагностики за допомогою відповідного значка

На сторінці вибору СТО розміщений перемикач режимів (список чи мапа) також із невеликим зображенням мапи для наглядності. При додаванні банківської картки користувач бачить повідомлення про те, що дані зашифровані, і спокійно заносит деталі у форму. Насамкінець, при успішному бронюванні водій переходить на відповідну сторінку і бачить деталі та знімок мапи, де знаходиться СТО. Це допомагає зафіксувати дані про місце та час візиту для виконання сервісних робіт.

4.3. Особливості тестування

Особливістю розробленого програмного забезпечення є багатокомпонентність, тому кожну із частин необхідно було тестувати різними методами. Для модулів системи було проведено unit тестування, чого загалом достатньо для того, щоб переконатися у правильності роботи. Загалом метою unit тестування було більше попередження виникнення помилок на головному сервері, адже кожного разу при запуску процесу CI у тестовому середовищі поетапно виконуються тести і якщо якісь частини коду не пройдуть перевірку нова збірка проєкту не встановится на сервер.

На стороні клієнтської частини все набагато складніше, адже необхідно тестувати як функції та методи, так і наявність тих чи інших елементів на сторінках та, насамкінець, стильове оформлення та регресію. З метою тестування функцій класів було використано вбудований в Angular інструмент karma та jasmine. Для всіх сервісів і ріре були створені ізольовані тести, в яких створювалися фікстури, що передавалися в обробку і результат порівнювався із очікуваним.

Більш важливим етапом тестування було e2e тестування, що передбачало перевірки усіх можливих сценаріїв використання веб-додатку. Для цього необхідно симулювати роботу програмного забезпечення у реальних умовах, що будуть враховувати багато факторів, наприклад, погану якість інтернет з’єднання, помилку в номері авто чи поштовому

індексі користувача, ситуацію, коли користувач не завершив бронювання та багато інших. Цілком зрозуміло, що для тестування усіх сценаріїв необхідно багато людських ресурсів та часу, тому було прийняте рішення про автоматизацію тестування засобами puppeteer та Jest [19].

Бібліотека puppeteer призначена для автоматизації роботи з браузером Google Chrome. Зокрема, за допомогою puppeteer можна створювати програми для автоматичного збору даних з веб-сайтів або проводити тестування, імітуючи дії звичайного користувача. На рис. 4.11 зображено блок-схему, що ілюструє схему роботи puppeteer.

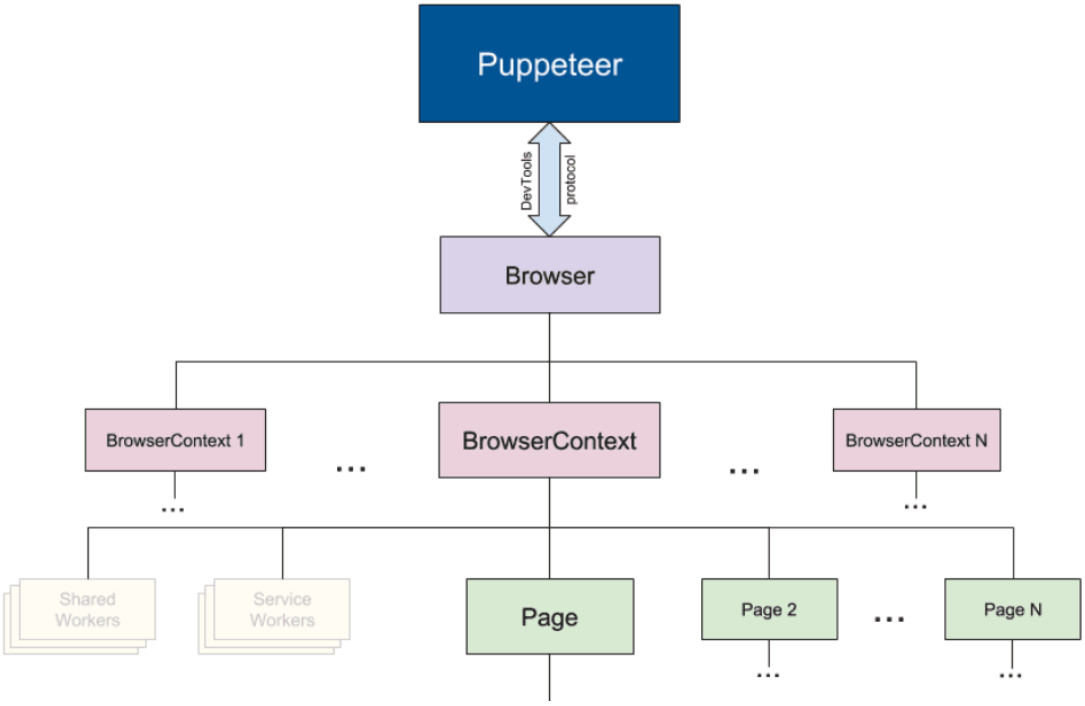


Рис. 4.11. Схема роботи puppeteer headless testing [20]

Для проведення e2e тестування необхідно описати найважливіші сценарії та знайти на кожній сторінці веб-додатку ключові елементи, відображення чи приховування яких є критичним. Було прийняте рішення протестувати процес реєстрації, додавання автомобіля та бронювання ТО. Сценарії тестування винесені у табл. 4.1.

Таблиця 4.1

Сценарії e2e тестування

№	Назва сценарію	Кроки сценарію	Очікувані результати
1	Реєстрація користувача	1. Перейти на головну сторінку додатку. 2. Натиснути на кнопку «log in». 3. На сторінці входу натиснути на кнопку «register». 4. Заповнити форму даними. 5. Натиснути кнопку «confirm»	У навігаційній панелі, що знаходиться вгорі веб-застосунку з’явилися додаткові посилання на сторінки із бронюваннями та інформацією про список автомобілів. Сторінка бронювань не містить жодного бронювання. Сторінка зі списком авто не містить жодного авто.
2	Додавання автомобіля з коректним номером	1. Натиснути на напис vehicles у навігаційній панелі, що знаходиться вгорі веб-застосунку. 2. Після переходу на відповідну сторінку натиснути кнопку «add vehicle». 3. Ввести державний номер авто. 4. Натиснути «пошук». 5. Авто знайдене, натиснути кнопку «так».	На сторінці зі списком автомобілів з'явилася інформація про автомобіль, який було додано, при цьому вказано правильний номер.
3	Додавання автомобіля з некоректним номером	1. Пункти 1-5 аналогічні сценарію №2. 2. Автомобіль не знайдений, закрити вікно пошуку.	На сторінці зі списком автомобілів відсутня інформація про транспортний засіб, оскільки дані з номерної таблички хибні.

Продовження табл. 4.2

4	Бронювання ТО	1. Перейти на головну сторінку. 2. Ввести дійсний номер автомобіля та поштовий індекс і натиснути кнопку «далі». 3. Вибрати необхідну категорію робіт. 4. Обрати необхідні сервіси. 5. Із списку доступних СТО поруч вибрати необхідну. 6. Додати банківську картку у платіжні методи та вказати її при оплаті. 7. Сплатити бронювання.	Користувач перенаправляється на сторінку із повною інформацією про бронювання – ціною, часом та місцем проведення робіт та знімком карти з маркером,що позначає розташування СТО.

Для того, щоб puppeteer міг доступитися до необхідних елементів необхідно вказати унікальні селектори, наприклад, класи. Оскільки у розробленому веб-додатку використовується інкапсуляція стилів – цей варіант став неможливим. Але була знайдена альтернатива – x-path, яки є своєрідним шляхом до необхідного елемента і не змінюється внаслідок модифікації стилів, а лише в разі зміни структури DOM дерева. Таким чином, при запуску puppeteer симулює реального користувача та заповнює всі відповідні поля наперед заданими даними і потім збирає інформацію про результат з потрібної сторінки. Jest у свою ж чергу порівнює вхідні параметри, очікуваний результат та отриманий і відповідно відмічає тест як пройдений чи не складений.

ВИСНОВКИ

Метою даного дипломного проєкту було розроблення веб-додатку для підтримки процесу технічного обслуговування автомобілів.

Використання сучасної технології SPA для розробки веб-додатка дозволило створити програмне забезпечення, що працює на будь-якій операційній системі та у більшості сучасних браузерів, не потребуючи значних обчислювальних потужностей. Саме швидкодія, надійність та кросплатформність відрізняють додатки, що розроблені за принципом SPA від інших аналогів. Також даний вид програмного забезпечення дозволяє виконувати розробку набагато швидше за рахунок додаткових пакетів, бібліотек та вбудованих у фреймворки інструментів.

Розроблене програмне забезпечення:

1. Дозволяє автоматизувати підтримку процесу технічного обслуговування автомобілів.
2. Пропонує найбільш актуальні продукти, залежно від пробігу автомобіля.
3. Мінімізує час очікування в чергах та додаткову комунікацію між СТО та власником транспортного засобу.
4. Має можливість інтегруватися із компаніями, що мають великі автопарки.
5. Має зрозумілий, інтуїтивний дизайн, що адаптується під пристрої із різною роздільною здатністю.
6. Надсилає сповіщення при оновленні статусу бронювання та за три дні до запланованого ТО.

Програмне забезпечення реалізоване у повному обсязі, функціональні та нефункціональні вимоги, що були поставлені до продукту виконані, програмне забезпечення є протестованим та стабільним.

Розроблене програмне забезпечення має більшу функціональність, ніж конкурентні рішення та інтуїтивно зрозумілий користувацький

інтерфейс. Крім того, інсує можливість інтеграцій компаній власників великих автопарків для використання даного веб-додатку.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

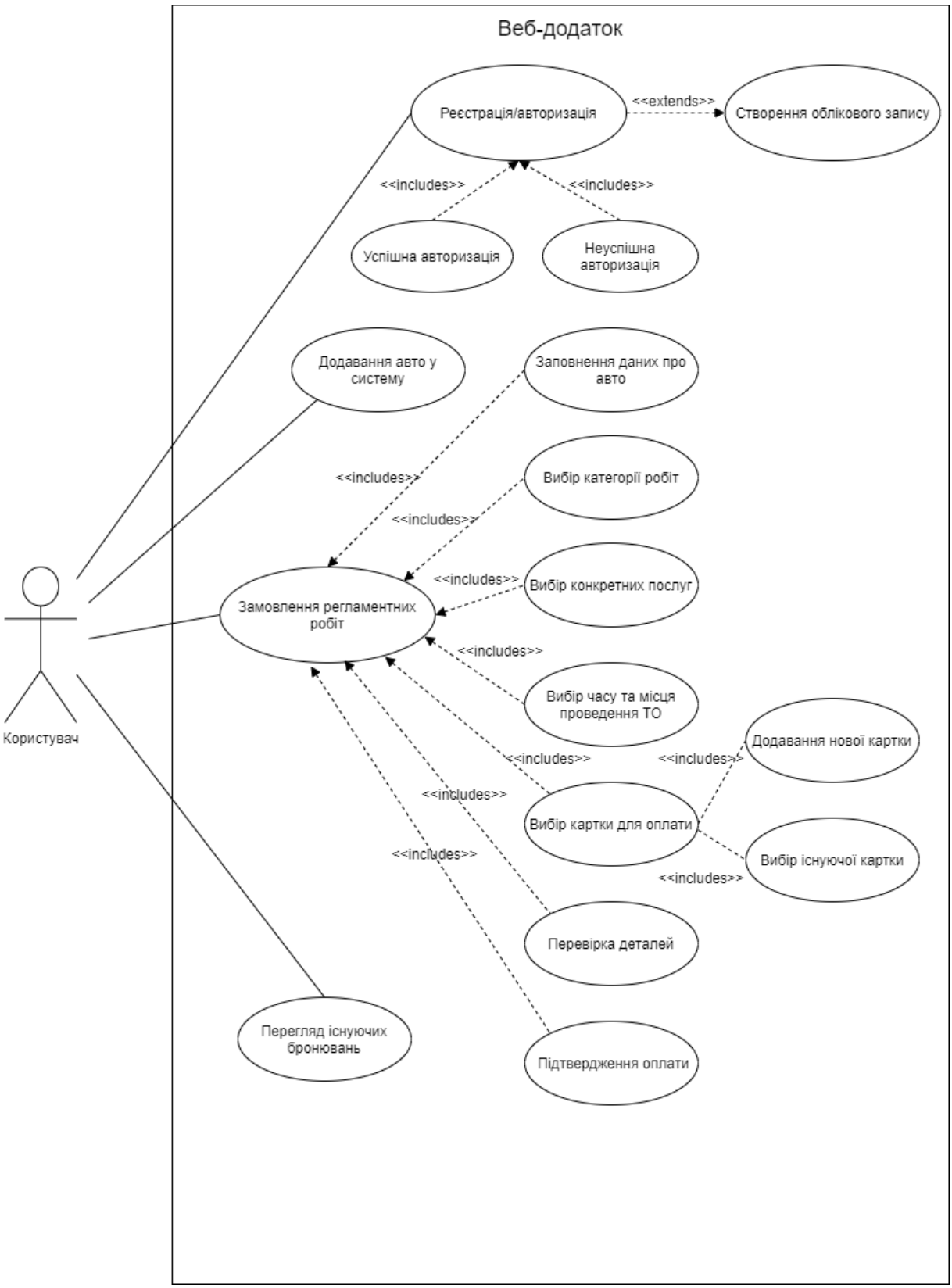
1. В Украине вырос уровень автомобилизации [Електроний ресурс] – Режим доступу: http://www.autoconsulting.com.ua/article.php?sid=35442.
2. Top 10 In-Demand programming languages to learn in 2020 [Електроний ресурс] – Режим доступу: https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e
3. 10 Best Java Frameworks to Use in 2020 [Електроний ресурс] – Режим доступу: https://hackr.io/blog/java-frameworks
4. Top 10 NodeJS Frameworks For Developers in 2020 [Електроний ресурс] – Режим доступу: https://codersera.com/blog/top-10-nodejs-frameworks-for-developers-in-2020/
5. When, How And Why Use Node.js as Your Backend [Електроний ресурс] – Режим доступу: https://www.netguru.com/blog/use-node-js-backend
6. A JavaScript library for building user interfaces [Електроний ресурс] – Режим доступу: https://reactjs.org/docs/getting-started.html
7. Introduction to the Angular Docs [Електроний ресурс] – Режим доступу: https://angular.io/docs
8. What is Vue.js? [Електроний ресурс] – Режим доступу: https://vuejs.org/v2/guide/
9. Reactive Extensions Library for JavaScript [Електроний ресурс] – Режим доступу: https://rxjs-dev.firebaseapp.com/guide/overview
10. Что такое Bootstrap? [Електроний ресурс] – Режим доступу: https://webformyself.com/chto-takoe-bootstrap/
11. MySQL [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/MySQL
12. PostgreSQL [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/PostgreSQL
13. PostgreSQL vs MySQL [Електроний ресурс] – Режим доступу: https://www.postgresqltutorial.com/postgresql-vs-mysql/

14. Software Design Principles DRY and KISS [Електроний ресурс] – Режим доступу: https://dzone.com/articles/software-design-principles-dry-and-kiss
15. My take on Redux architecture [Електроний ресурс] – Режим доступу: https://krasimirtsonev.com/blog/article/my-take-on-redux-architecture
16. Stripe.js Reference [Електроний ресурс] – Режим доступу: https://stripe.com/docs/js
17. Swagger Codegen Documentation [Електроний ресурс] – Режим доступу: https://swagger.io/docs/open-source-tools/swagger-codegen/
18. Why the UX Is Important for Your Business [Електроний ресурс] – Режим доступу: https://rubygarage.org/blog/why-the-ux-is-important-for-your-business
19. Jest и Puppeteer: автоматизация тестирования веб-интерфейсов [Електроний ресурс] – Режим доступу: https://habr.com/ru/company/ruvds/blog/342578/
20. Getting Started With the Puppeteer API for Headless Chrome [Електроний ресурс] – Режим доступу: https://dzone.com/articles/getting-started-with-the-puppeteer-api-for-headles

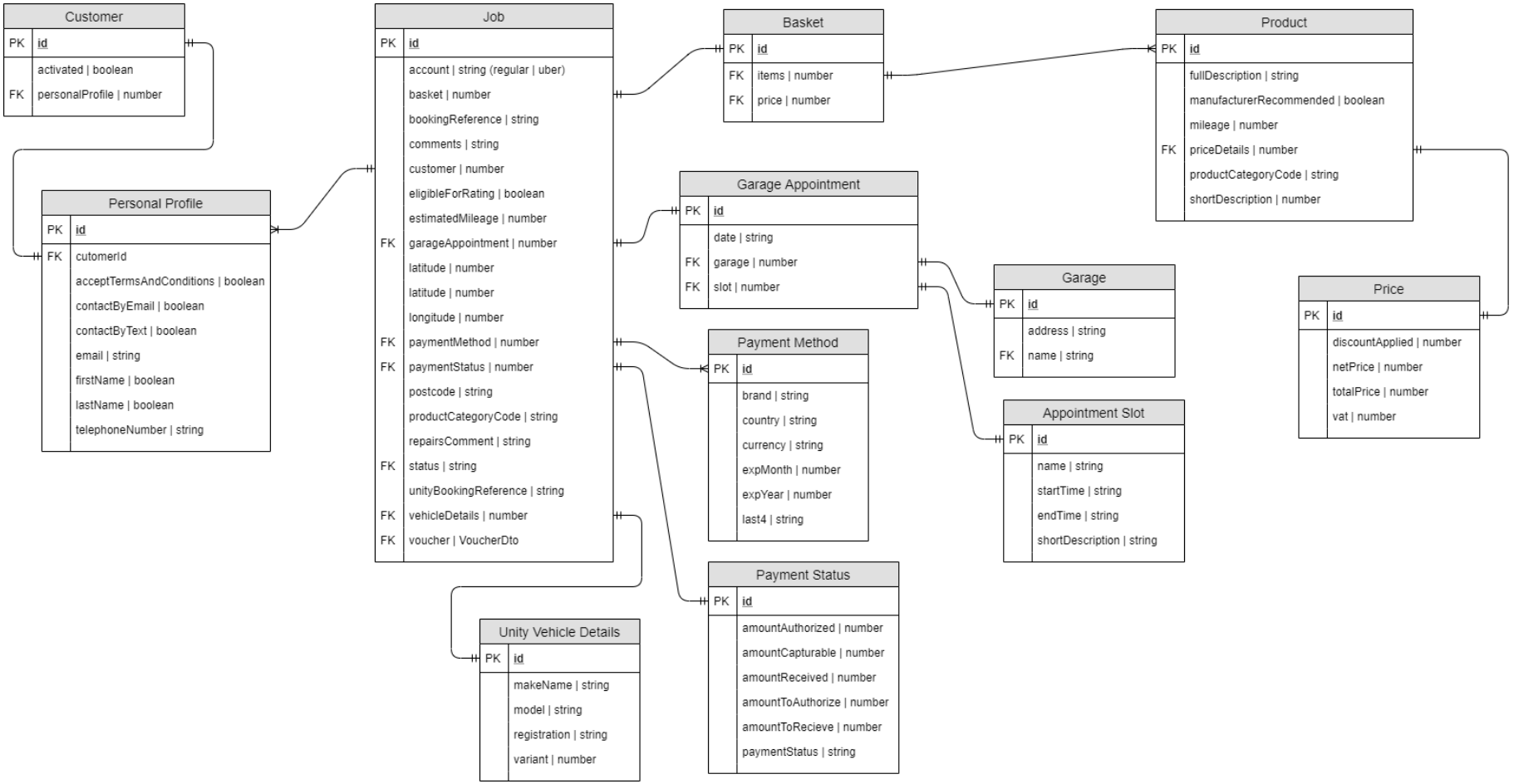
ДОДАТКИ

Додаток 1

Копії графічних матеріалів

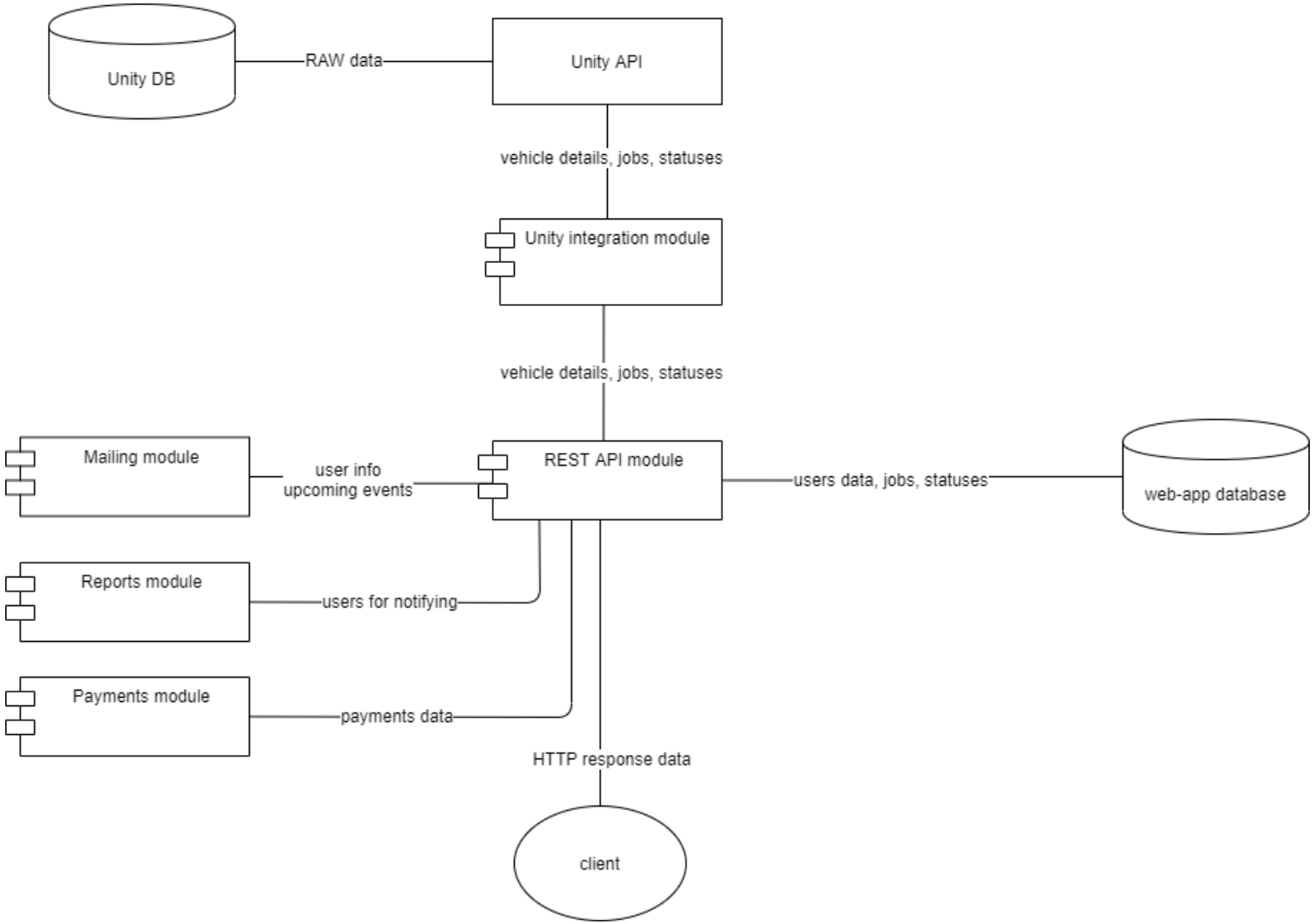


ДП.045440-06-99
Веб-додаток для підтримки процесу технічного обслуговування автомобілів. Функціональність програмних засобів.
UML-діаграма прецедентів



ДП.045440-07-99
Веб-додаток для підтримки процесу технічного обслуговування автомобілів. Структура бази даних. ER-діаграма

Схема взаємодії програмних модулів



Телефус І.А., група КП-61

Аналізатор розміру скомпільованих компонентів клієнтської частини веб-додатку

Телефус І.А., група КП-61

Додаток 2

Лістинг модулю вибору часу і місця проходження сервісного обслуговування

availability-date-picker-large.component.html

```
<div class="row py-4 ">
    <div class="col-lg-12 px-lg-6 py-2">
        <h2 class="heading-title">Appointment</h2>
    </div>
</div>
<div class="row">
    <div class="col-lg-12 px-lg-6">
        <div class="data-control">
            <div class="data-prev">
                <a class=" mycar-icon mycar-angle-
left" href="javascript:void(0)" (click)="garageAppointmentService.previous()" [c
lass.active]="garageAppointmentService.hasPrevious()"></a>
            </div>
            <div class="data-next">
                <a class="mycar-icon mycar-angle-
right" href="javascript:void(0)" (click)="garageAppointmentService.next()" [clas
s.active]="garageAppointmentService.hasNext()"></a>
            </div>
        </div>
        <div class="data-wrap d-flex justify-content-between">
            <div class="data-item bg-
white" *ngFor="let day of garageAppointmentService.page.content; let i = index"
[class.disabled]="!day.available">
                <div class="d-flex flex-column text-center px-4">
                    <h5 class="calendar-selector" [ngClass]="{'day-
selected': garageAppointmentService.isDaySelected(day)}">
                        {{day.date | date:'EE' }}</h5>
                    <h6 class="calendar-selector-
day">{{day.date | date:'d LLLL' }}</h6>
                </div>
                <ul class="data-tabs d-flex flex-column text-center">
                    <ng-
container *ngFor="let slot of day.slots; let j = index">
                        <li *ngIf="slot.available" (click)="garageAppointm
entService.selectDay(day, j)" [class.active]="garageAppointmentService.isSlotInd
exSelected(day, j)">
                            <p class="body-small mb-0">{{slot.name}}</p>
                        </li>
                        <li class="disabled" *ngIf="!slot.available">
                            <p class="body-small mb-0">Not available</p>
                        </li>
                    </ng-container>
                </ul>
            </div>
        </div>
    </div>
</div>
```

availability-date-picker-large.component.scss

```
.data-wrap {
    margin: 0 -0.1rem;
}
.data-control {
    display: relative;
    width: 100%;
}
.data-prev,
```

```
.data-next {
    position: absolute;
    top: 0;
    overflow: hidden;
    height: 5.4rem;
    width: 3.7rem;
}
.data-prev {
    left: 0;
}
.data-next {
    right: 0;
}
.data-prev .mycar-icon.active,
.data-next .mycar-icon.active {
    color: #07828d;
}
.data-prev .mycar-icon,
.data-next .mycar-icon {
    text-decoration: none;
    font-size: 3.3rem;
    padding-top: 0.9rem;
    background: #fff;
    color: #c7c8ca;
    height: 5.4rem;
    width: 5.4rem;
}
.data-prev .mycar-icon {
    border-radius: 50% 0 0 50%;
    padding-left: 1.5rem;
    text-align: left;
}
.data-next .mycar-icon {
    border-radius: 0 50% 50% 0;
    margin-right: 0;
    float: right;
    padding-left: 1rem;
}
.data-item {
    width: 20%;
    margin: 0 0.1rem;
}
.calendar-selector {
    margin-top: 1.8rem;
    line-height: 0.1rem;
}
.calendar-selector-day {
    margin-bottom: 1rem;
}
ul {
    padding: 0 1rem;
}
li {
    border-radius: 2rem;
    font-weight: 600;
    padding: 0.5rem;
    color: #07828d;
    margin-bottom: 0.7rem;
}
li:hover {
    cursor: pointer;
```

```
}
li.active {
    background: #07828d;
    color: #fff;
}
.day-selected {
    font-family: NewTransport-Bold, sans-serif;
}
li.disabled,
.disabled {
    color: #a7a9ac !important;
    background: none;
    font-weight: 400;
    cursor: auto;
}
```

availability-date-picker-large.component.ts

```
import { Component, OnInit } from '@angular/core';
import { GarageAppointmentService } from '../garage-appointment.service';

@Component({
  selector: 'app-availability-date-picker-large',
  templateUrl: './availability-date-picker-large.component.html',
  styleUrls: ['./availability-date-picker-large.component.scss']
})
export class AvailabilityDatePickerLargeComponent implements OnInit {

  constructor( public garageAppointmentService: GarageAppointmentService )
{ }

  ngOnInit() {
  }
}
```

garage-map.service.ts

```
import { Injectable } from '@angular/core';
import { GarageAppointmentService, AvailabilityDateExtendedDto } from
'../garage-appointment.service';
import { ForDateGarageAndSlotsDto, ForDateAvailabilityDto } from
'src/app/api';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class GarageMapService {
  garagesByDateLength: number;
  private garagesByDay: ForDateGarageAndSlotsDto[] = [];
  private _garages: {
    [key: string]: ForDateAvailabilityDto;
  } = {};
  public set garages(value: {
    [key: string]: ForDateAvailabilityDto;
  }) {
    this._garages = value;
    this.resetGarages(this.garageAppointmentService.currentDay);
  }
  public garagesByDayObservable: BehaviorSubject<{ [key: number]:
ForDateGarageAndSlotsDto }> = new BehaviorSubject(this.garagesByDay);
```

```
        private currentGarageId: number;
        private _currentSelectedGarage: ForDateGarageAndSlotsDto;
        public get currentSelectedGarage(): ForDateGarageAndSlotsDto {
          return this._currentSelectedGarage;
        }
        public set currentSelectedGarage(value: ForDateGarageAndSlotsDto) {
          this._currentSelectedGarage = value;
          let newSlotId = null;
          const foundSlot = value.slots.find(slot => slot.available);
          if (foundSlot) {
            newSlotId = foundSlot.id;
          }

this.garageAppointmentService.selectCurrentGarageAndSlot(value.garage.id,
newSlotId);
        }

        constructor(
          private garageAppointmentService: GarageAppointmentService
        ) {

this.garageAppointmentService.currentDayObservable.subscribe(currentDay       =>
this.resetGarages(currentDay));

this.garageAppointmentService.currentGarageObservable.subscribe(currentGarage =>
{
            this.currentGarageId = null;
            if (currentGarage) {
              this.currentGarageId = currentGarage.id;
            }
            this.setCurrentGarage();
          });
        }
        private resetGarages(currentDay: AvailabilityDateExtendedDto) {
          this.garagesByDateLength = 0;
          this.garagesByDay = [];
          if (currentDay && this._garages[currentDay.date]) {
            this.garagesByDateLength                                          =
this._garages[currentDay.date].garages.length;
            this.garagesByDay = this._garages[currentDay.date].garages;
          }
          this.garagesByDayObservable.next(this.garagesByDay);
          this.setCurrentGarage();
        }
        private setCurrentGarage() {
          this._currentSelectedGarage = null;
          if (this.currentGarageId && this.garagesByDay.length > 0) {
            this._currentSelectedGarage  =  this.garagesByDay.find(garageAndSlots
=> garageAndSlots.garage.id === this.currentGarageId);
          }
        }

      }
```

garages-map.component.html

```
      <div #mapContainer id="map" [style.height.px]="height" class="row bg-white
pt-4"></div>
```

garages-map.component.scss

```
      * {
```

```
  cursor: pointer;
}
```

garages-map.component.html

```
import { Component, OnInit, AfterViewInit, ViewChild, ElementRef, OnDestroy
} from '@angular/core';
import { ForDateGarageAndSlotsDto } from 'src/app/api';
import { GarageAppointmentService } from '../garage-appointment.service';
import { JobStateService } from '../../job-state.service';
import { GarageMapService } from './garage-map.service';
import { ParametersService, PARAMETER_NAME } from
'src/app/commons/parameters/parameters.service';
import { MapStyle } from './map.style';


@Component({
  selector: 'app-garages-map',
  templateUrl: './garages-map.component.html',
  styleUrls: ['./garages-map.component.scss']
})
export class GaragesMapComponent implements AfterViewInit, OnInit,
OnDestroy {
  @ViewChild('mapContainer') gmap: ElementRef;

  height: number = window.innerHeight - 344;
  iconBasePath: string;
  map: google.maps.Map;
  mapStyles: MapStyle = new MapStyle();
  mapOptions: google.maps.MapOptions = {
    zoom: 10,
    disableDefaultUI: true,
    zoomControl: true,
    zoomControlOptions: {
      position: google.maps.ControlPosition.TOP_RIGHT
    },
    styles: []
  };
  private markers: {
    [key: number]: google.maps.Marker;
  } = {};

  constructor(
    private garageAppointmentService: GarageAppointmentService,
    private jobStateService: JobStateService,
    private garageMapService: GarageMapService,
    private parametersService: ParametersService
  ) { }

  ngOnInit() {
    this.iconBasePath =
this.parametersService.get(PARAMETER_NAME.customer_app_base_path) +
'/assets/icons';
  }
  cleanMarkers() {
    Object.values(this.markers).forEach((marker) => marker.setMap(null));
    this.markers = {};
  }
  showMarkers(garagesByDay: { [key: number]: ForDateGarageAndSlotsDto }):
void {
    this.cleanMarkers();
    Object.values(garagesByDay).forEach(garageAndSlots => {
```

```
            const coordinates = new google.maps.LatLng(
              garageAndSlots.garage.address.latitude,
              garageAndSlots.garage.address.longitude
            );
            const marker = new google.maps.Marker({
              position: coordinates,
              map: this.map,
              icon: `${this.iconBasePath}/garageAvailable.png`,
              optimized: false,
            });
            marker.set('garageId', garageAndSlots.garage.id);
            this.addClickEvtToMarker(marker, garageAndSlots);
            this.markers[garageAndSlots.garage.id] = marker;
          });
        }

        addClickEvtToMarker(marker:     google.maps.Marker,     garageAndSlots:
ForDateGarageAndSlotsDto): void {
          google.maps.event.addListener(marker, 'click', () => {
            this.garageMapService.currentSelectedGarage = garageAndSlots;
          });
        }
        selectMarkerForGarage(garageId: number) {
          Object.values(this.markers).forEach(marker                              =>
marker.setIcon(`${this.iconBasePath}/garageAvailable.png`));
          if (garageId) {

this.markers[garageId].setIcon(`${this.iconBasePath}/garageSelected.png`);

          }
        }

        mapInitializer(): void {
          for (const style of this.mapStyles.config) {
            this.mapOptions.styles.push(style);
          }
          this.mapOptions.center = new google.maps.LatLng(
            this.jobStateService.job.latitude,
            this.jobStateService.job.longitude
          );
          this.map      =      new      google.maps.Map(this.gmap.nativeElement,
this.mapOptions);
          // tslint:disable-next-line: no-unused-expression
          new google.maps.Marker({
            position: this.mapOptions.center,
            map: this.map,
            icon: `${this.iconBasePath}/icon-target.png`,
            optimized: false
          });
        }

        ngAfterViewInit() {
          this.mapInitializer();
          this.garageMapService.garagesByDayObservable.subscribe((garagesByDay)
=> {
            this.showMarkers(garagesByDay);
          });

this.garageAppointmentService.currentGarageObservable.subscribe(currentGarage =>
{
            this.selectMarkerForGarage(currentGarage ? currentGarage.id : null);
```

```
    });
  }
  /** @internal */
  ngOnDestroy() {
    // unsubscribe all registered observable subscriptions
    // remove all listeners from the map instance
    google.maps.event.clearInstanceListeners(this.map);
  }
}
```

map.style.ts

```
export class MapStyle {
    public config: google.maps.MapTypeStyle[] = [
        {
            'elementType': 'geometry',
            'stylers': [
                {
                    'color': '#f5f5f5'
                }
            ]
        },
        {
            'elementType': 'labels.icon',
            'stylers': [
                {
                    'visibility': 'off'
                }
            ]
        },
        {
            'elementType': 'labels.text.fill',
            'stylers': [
                {
                    'color': '#616161'
                }
            ]
        },
        {
            'elementType': 'labels.text.stroke',
            'stylers': [
                {
                    'color': '#f5f5f5'
                }
            ]
        },
        {
            'featureType': 'administrative.land_parcel',
            'elementType': 'labels.text.fill',
            'stylers': [
                {
                    'color': '#bdbdbd'
                }
            ]
        },
        {
            'featureType': 'poi',
            'elementType': 'geometry',
            'stylers': [
                {
                    'color': '#eeeeee'
                }
```

```
    ]
},
{
    'featureType': 'poi',
    'elementType': 'labels.text.fill',
    'stylers': [
        {
            'color': '#757575'
        }
    ]
},
{
    'featureType': 'poi.park',
    'elementType': 'geometry',
    'stylers': [
        {
            'color': '#e5e5e5'
        }
    ]
},
{
    'featureType': 'poi.park',
    'elementType': 'labels.text.fill',
    'stylers': [
        {
            'color': '#9e9e9e'
        }
    ]
},
{
    'featureType': 'road',
    'elementType': 'geometry',
    'stylers': [
        {
            'color': '#ffffff'
        }
    ]
},
{
    'featureType': 'road.arterial',
    'elementType': 'labels.text.fill',
    'stylers': [
        {
            'color': '#757575'
        }
    ]
},
{
    'featureType': 'road.highway',
    'elementType': 'geometry',
    'stylers': [
        {
            'color': '#dadada'
        }
    ]
},
{
    'featureType': 'road.highway',
    'elementType': 'labels.text.fill',
    'stylers': [
        {
```

```
                    'color': '#616161'
                }
            ]
        },
        {
            'featureType': 'road.local',
            'elementType': 'labels.text.fill',
            'stylers': [
                {
                    'color': '#9e9e9e'
                }
            ]
        },
        {
            'featureType': 'transit.line',
            'elementType': 'geometry',
            'stylers': [
                {
                    'color': '#e5e5e5'
                }
            ]
        },
        {
            'featureType': 'transit.station',
            'elementType': 'geometry',
            'stylers': [
                {
                    'color': '#eeeeee'
                }
            ]
        },
        {
            'featureType': 'water',
            'elementType': 'geometry',
            'stylers': [
                {
                    'color': '#c9c9c9'
                }
            ]
        },
        {
            'featureType': 'water',
            'elementType': 'labels.text.fill',
            'stylers': [
                {
                    'color': '#9e9e9e'
                }
            ]
        }
    ];
}
```

garage-appointment.service.ts

```
import { Injectable } from '@angular/core';
// tslint:disable-next-line:max-line-length
import { AvailabilityDateDto, AvailabilitySlotAndGaragesDto,
ForPeriodAvailabilityResponseDto, GarageDetailsDto } from '../../api';
import { ScreenSizeService } from 'src/app/commons/screen-size.service';
import { BehaviorSubject } from 'rxjs';
```

```
@Injectable({
  providedIn: 'root'
})
export class GarageAppointmentService {
  public page: {
    content: Array<AvailabilityDateExtendedDto>,
    number: number,
    size: number,
    total: number
  } = {
      content: [],
      number: 0,
      size: 1,
      total: 0
    };
  private contentToScroll: Array<AvailabilityDateExtendedDto> = [];
  private _currentGarage: GarageDetailsDto;
  private _currentDay: AvailabilityDateExtendedDto;
  public currentSlot: AvailabilitySlotAndGaragesDto;
  private _currentSlotIndex: number;
  public availableGaragesCount = 0;
  public totalGaragesCount = 0;
  public currentGarageAvailable = false;
  public currentGarageObservable: BehaviorSubject<GarageDetailsDto> = new
BehaviorSubject(null);
  public currentDayObservable:
BehaviorSubject<AvailabilityDateExtendedDto> = new
BehaviorSubject(this._currentDay);
  public currentSlotIndexObservable: BehaviorSubject<number> = new
BehaviorSubject(null);

  constructor(private screenSizeService: ScreenSizeService) {
  }

  public get currentDay(): AvailabilityDateExtendedDto {
    return this._currentDay;
  }
  public set currentDay(value: AvailabilityDateExtendedDto) {
    this._currentDay = value;
    if (!value || !value.available) {
      // we also mark as currentGarage is not available
      this.currentGarageAvailable = false;
    }
    this.currentDayObservable.next(value);
  }
  public get currentGarage(): GarageDetailsDto {
    return this._currentGarage;
  }
  public set currentGarage(value: GarageDetailsDto) {
    this._currentGarage = value;
    this.currentGarageObservable.next(value);
    if (value) {
      this.currentGarageAvailable = true;
    } else {
      this.currentGarageAvailable = false;
    }

  }
  public get currentSlotIndex(): number {
    return this._currentSlotIndex;
```

```
        }
        public set currentSlotIndex(value: number) {
          this._currentSlotIndex = value;
          this.currentSlotIndexObservable.next(value);
        }
        public isLoaded() {
          return this.contentToScroll.length > 0;
        }
        public load(response: ForPeriodAvailabilityResponseDto) {
          const availabilityList = response.content;
          if (this.screenSizeService.isLarge) {
            // large screen the current batch is 5
            this.page.size = 5;
            // also append fake days so that we are ok with UI
            // "The following 5-day slot is displayed for as long as the last day
of the 'next 5 days' is not after
            // the last available date from the availability API; if that's the
case,
            // then the last 5 days of the availability API are displayed)."
            const missingDays = this.page.size - response.content.length %
this.page.size;
            if (missingDays < this.page.size) {
              for (let i = 0; i < missingDays; i++) {
                const missingDay = new
Date(response.content[response.content.length - 1].date);
                missingDay.setDate(missingDay.getDate() + 1);
                availabilityList.push({ available: false, date:
missingDay.toISOString(), slots: [{ available: false }, { available: false }] });
              }
            }
          } else {
            this.page.size = 1;
          }
          this.contentToScroll = availabilityList.map((availabilityDateDto,
index) => Object.assign({ index }, availabilityDateDto));
          this.page.number = 0;
          this.page.total = this.contentToScroll.length / this.page.size;
          if (this.currentDay) {
            // we have something selected
            this.syncPageWithcurrentDay();
            this.computeSelectedGarage();
          } else {
            this.setPageData(0);
          }

        }
        public hasNext() {
          return this.page.number < this.page.total - 1;
        }
        public hasPrevious() {
          return this.page.number > 0;
        }
        public next() {
          this.setPageData(this.page.number + 1);
        }
        public previous() {
          this.setPageData(this.page.number - 1);
        }
        private setPageData(newPageNumber: number, autoSelectAvailability = true)
{
          if (newPageNumber >= 0 && newPageNumber < this.page.total) {
```

```
            // valid newPageIndex
            // page start index
            const pageOffset = newPageNumber * this.page.size;
            // get the slice of data
            this.page.content = this.contentToScroll.slice(pageOffset, pageOffset
+ this.page.size);
            this.page.number = newPageNumber;
            if (autoSelectAvailability) {
              this.selectDay();
            }
          }
        }
        public  selectDay(newDay?:  AvailabilityDateExtendedDto,  newSlotIndex?:
number) {
          if (!newDay) {
            // just find the first available in the current page
            for (const day of this.page.content) {
              if (day.available) {
                newDay = day;
                break;
              }
            }
          }
          if (!newDay) {
            // no available one was found so we just select the first one in the
page
            newDay = this.page.content[0];
          }
          if (newDay) {
            this.currentDay = this.contentToScroll[newDay.index];
            if (this.currentDay.available) {
              // if current day is available we also select the time slot
              // if not we don't change so that when we get to an available on we
also have the slot
              if (!(newSlotIndex >= 0)) {
                // next select next available slot for the day
                newSlotIndex = -1;
                if (this.currentSlotIndex >= 0) {
                  // we try to keep the current one selected
                  if  (this.currentSlotIndex  <  this.currentDay.slots.length  &&
this.currentDay.slots[this.currentSlotIndex].available) {
                    newSlotIndex = this.currentSlotIndex;
                  }
                }
                if (!(newSlotIndex >= 0)) {
                  for (let j = 0; j < this.currentDay.slots.length; j++) {
                    if (this.currentDay.slots[j].available) {
                      newSlotIndex = j;
                      break;
                    }
                  }
                }
              }
            }
            this.selectSlot(newSlotIndex);
            this.syncPageWithcurrentDay();
            return true;
          }
          return false;
        }
        public selectSlot(newSlotIndex: number) {
```

```
          if (newSlotIndex >= 0 && newSlotIndex < this.currentDay.slots.length) {
            this.currentSlotIndex = newSlotIndex;
            this.currentSlot = this.currentDay.slots[this.currentSlotIndex];
          } else {
            // we don't reset this.currentSlotIndex
            this.currentSlot = null;
          }
          this.computeSelectedGarage();
          this.computeGaragesCount();
        }
        public isSlotIndexSelected(day: AvailabilityDateExtendedDto, slotIndex:
number) {
          return this.currentDay && day.index === this.currentDay.index &&
slotIndex === this.currentSlotIndex;
        }
        public getAggregateAvailability() {
          return this.contentToScroll;
        }
        private syncPageWithcurrentDay() {
          // check and see that currentDay is in the current page
          if (this.currentDay) {
            const pageOffset = this.page.number * this.page.size;
            if (!(this.currentDay.index >= pageOffset && this.currentDay.index <
pageOffset + this.page.size)) {
              this.setPageData(Math.floor(this.currentDay.index                /
this.page.size), false);
            }
          }
        }
        private computeGaragesCount() {
          if (this.currentSlot && this.currentSlot.available) {
            this.totalGaragesCount = this.currentSlot.garages.length;
            this.availableGaragesCount = this.currentSlot.garages.filter((x) =>
x.available).length;
          } else {
            this.totalGaragesCount = 0;
            this.availableGaragesCount = 0;
          }
        }
        private computeSelectedGarage() {
          let found = false;
          if (this.currentGarage && this.currentSlot) {
            // we try to keep it
            for (const garage of this.currentSlot.garages) {
              if (garage.garage.id === this.currentGarage.id) {
                if (garage.available) {
                  found = true;
                  // we also update the whole object as these are not actualy same
references
                  this.currentGarage = garage.garage;
                }
                break;
              }
            }
          }
          if (!found) {
            // we keep this.currentGarage but we mark it as unavailable
            this.currentGarageAvailable = false;
          }
        }
```

```
        public isDaySelected(day: AvailabilityDateExtendedDto) {
          return this.currentDay && day.index === this.currentDay.index;
        }

        public reset(): void {
          this.currentDay = null;
          this.currentSlot = null;
          this.currentSlotIndex = undefined;
          this.currentGarage = null;
          this.contentToScroll = [];
        }

        public selectCurrentGarageAndSlot(newGarageId: number, newSlotId: string)
{
          if (newSlotId) {
            this.currentSlotIndex   =   this.currentDay.slots.findIndex(slot   =>
slot.id === newSlotId);
            this.currentSlot = this.currentDay.slots[this.currentSlotIndex];
          } else {
            this.currentSlotIndex = -1;
            this.currentSlot = null;
          }
          if (this.currentSlot) {
            for (const garage of this.currentSlot.garages) {
              if (garage.garage.id === newGarageId) {
                if (garage.available) {
                  this.currentGarage = garage.garage;
                }
                break;
              }
            }
          }
          this.computeGaragesCount();
        }
      }

      export interface AvailabilityDateExtendedDto extends AvailabilityDateDto {
        index: number;
      }
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБ-ДОДАТОК ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Виконав: Телефус Ілля Анатолійович

Науковий керівник: ст. викл. Гадиняк Руслан Анатолійович

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: розробити веб-додаток для підтримки процесу технічного обслуговування автомобілів, що за допомогою зручного інтерфейсу мінімізує витрати часу на комунікацію спеціалістів СТО та власників авто.

Завдання:

1. Проаналізувати галузь застосування та існуючі аналоги.
2. Розробити програмне забезпечення для підтримки процесу технічного обслуговування автомобілів.
3. Протестувати розроблене програмне забезпечення.

АКТУАЛЬНІСТЬ

1. Програмне забезпечення для власників авто:
   - збереження часу при виборі регламентних робіт, місця та часу проведення;
   - мінімізація комунікацій та зменшення імовірності помилок;
   - перегляд статусу замовлення онлайн.
2. Застосування розробленого веб-додатку в якості заміни аналогам, що не мають достаньої функціональності.

АНАЛІЗ НАЯВНИХ РІШЕНЬ

- Аналоги:
  - my.eurocar.com.ua;
  - oiler.ua;
  - autobooking.com.

- Недоліки:
  - необхідність телефонного дзвінка для підтвердження;
  - неочевидний інтерфейс користувача;
  - відсутність опцій із додавання в акаунт декількох авто.

ВИМОГИ ДО ФУНКЦІОНАЛЬНОСТІ

- постановка автомобілів для технічного обслуговування;
- підбір необхідних та планових регламентних робіт, зважаючи на інформацію про авто;
- пошук СТО, що знаходяться поряд з користувачем та відображення їх на мапі у разі наявності там вільних місць;
- розширення за рахунок інтеграції з компаніями, що мають великі автопарки та хочуть автоматизувати процеси з ТО автомобілів;
- перегляд бронювань та статусу бронювань ТО;
- здійснення оплати онлайн;
- реєстрація користувачів у системі.

ЗАСОБИ РЕАЛІЗАЦІЇ

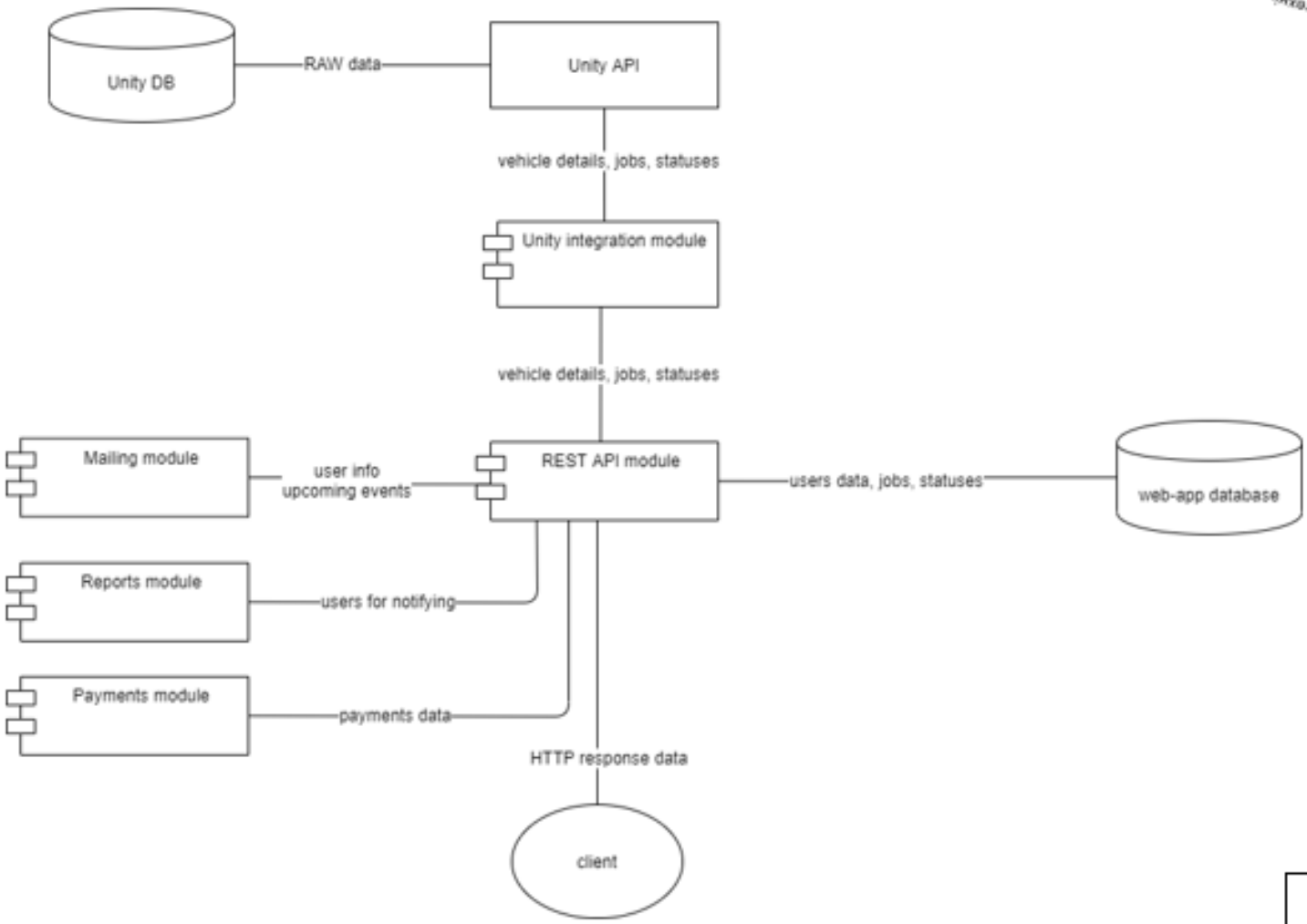
- Мова програмування **Java**.
- Серверний фреймворк **Spring boot**.
- Реляційна база даних **MySQL**.
- Веб-фреймворк **Angular8**.
- Бібліотека **Stripe.js** для обробки платежів.
- **Google Maps API** для відображення наявних СТО на мапі.
- Сервіс для розсилки повідомлень **Mailjet**.
- **Unity CRM API** для отримання даних про авто та їх власників.

АРХІТЕКТУРА ТА КОМПОНЕНТИ СИСТЕМИ

- Типи модулів:
  - модуль підбору сто поблизу;
  - модуль авторизації;
  - поштовий модуль;
  - модуль оплати.

- Користувачі:
  - власники автомобілів;
  - компанії партнери.

АРХІТЕКТУРА СИСТЕМИ

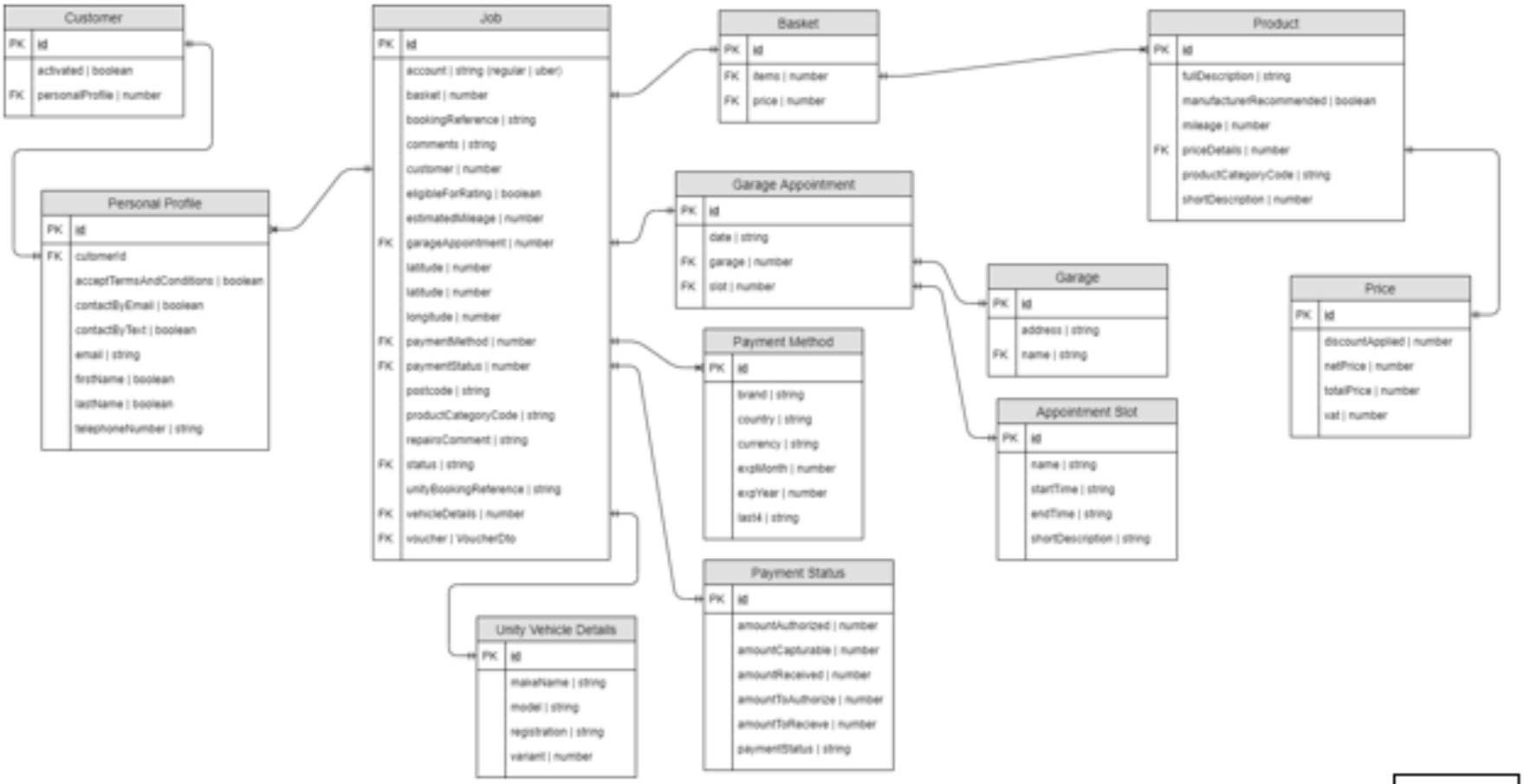


БАЗА ДАНИХ

- Реляційна база даних MySQL.
- Проведено першу, другу та третю нормалізації.
- Індекси для швидшого пошуку даних.

- Основні таблиці:
  - дані про користувачів: Customer, PersonalProfile;
  - дані про авто: UnityVehicleDetails;
  - фінансові дані: Price, PaymentMethod, PaymentStatus;
  - дані про замовлення: Job, Product, Basket, GarageAppointment, Garage, AppointmentSlot.

СТРУКТУРА БАЗИ ДАНИХ



РОЛІ КОРИСТУВАЧІВ ВЕБ-ДОДАТКУ

- Власник автомобіля
  - Вхід у систему.
  - Прив’язка автомобілів до облікового запису.
  - Замовлення регламентних робіт.
  - Перегляд статусу виконання замовлення.
  - Завантаження звіту виконаних послуг.
  - Залишення відгуку.
- Компанії партнери (опціонально)
  - Використання Unity CRM API для додавання інформації в систему.
  - Надання власного API для інтеграції із системою.

РОЛІ КОРИСТУВАЧІВ СИСТЕМИ

ТЕСТУВАННЯ

- Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам.
- Використовуються наступні методи:

  Нефункціональне тестування:

  - Тестування надійності.
  - Тестування роботи при навантаженні.
  - Тестування зручності використання.

  Функціональне тестування:

  - E2e тестування.
  - Мануальне тестування інтерфейсу.
  - Функціональне тестування точок системи.
  - Юніт-тестування функціональності сервера.

ОЦІНКА РОЗРОБЛЕНОГО ВЕБ-ДОДАТКА

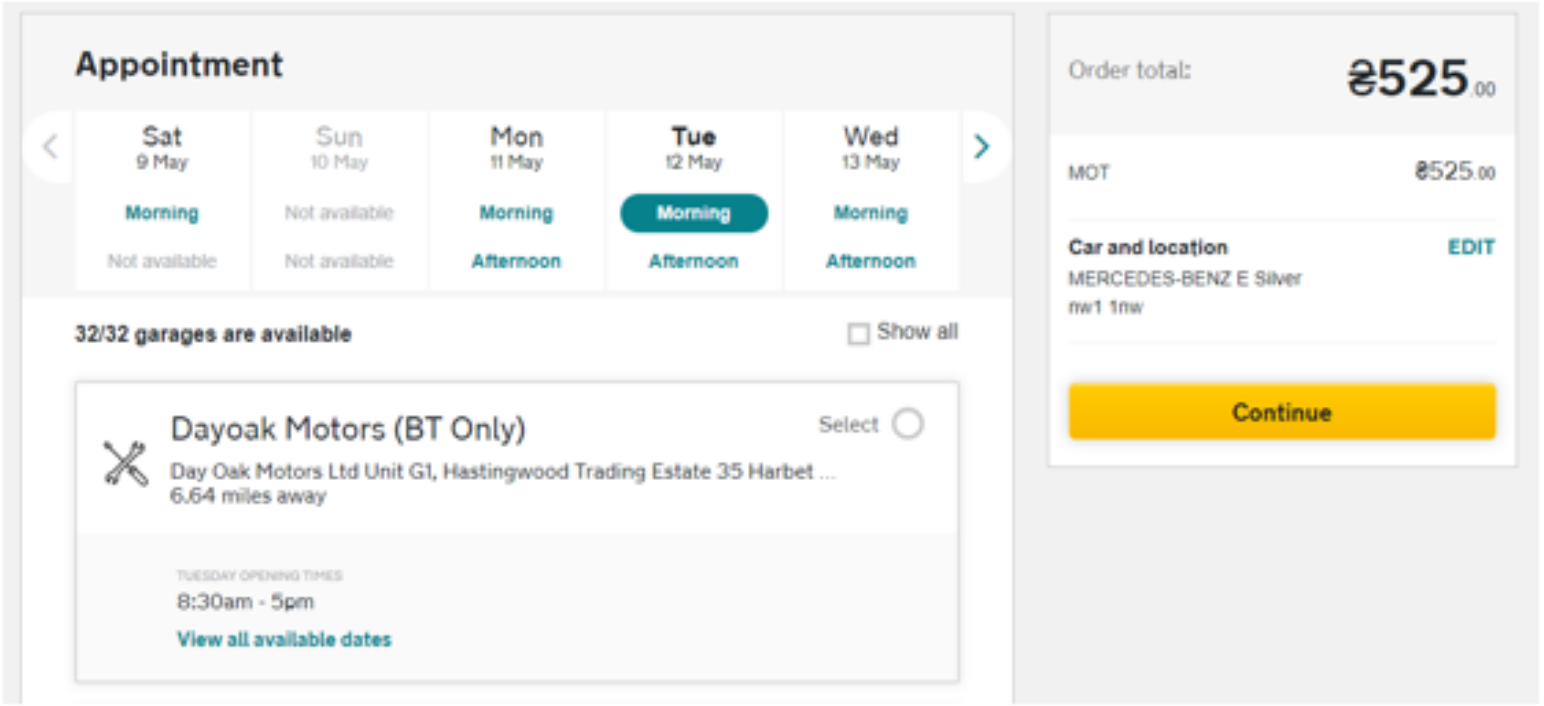
- Масштабованість ситеми.
- Безпека:
  - Застосування JWT токенів для авторизації користувачів;
  - ліміт кількості запитів до серверу;
  - безпечний формат запитів до бази даних;
  - валідація даних клієнтською та серверною частинами;
  - санітизація даних та захист від XSS атак.
- Надійність
  - Дані, що містяться у Unity CRM є перевіреними, а СТО - акредитованими.
- Швидкодія

ПЕРЕВАГИ РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ

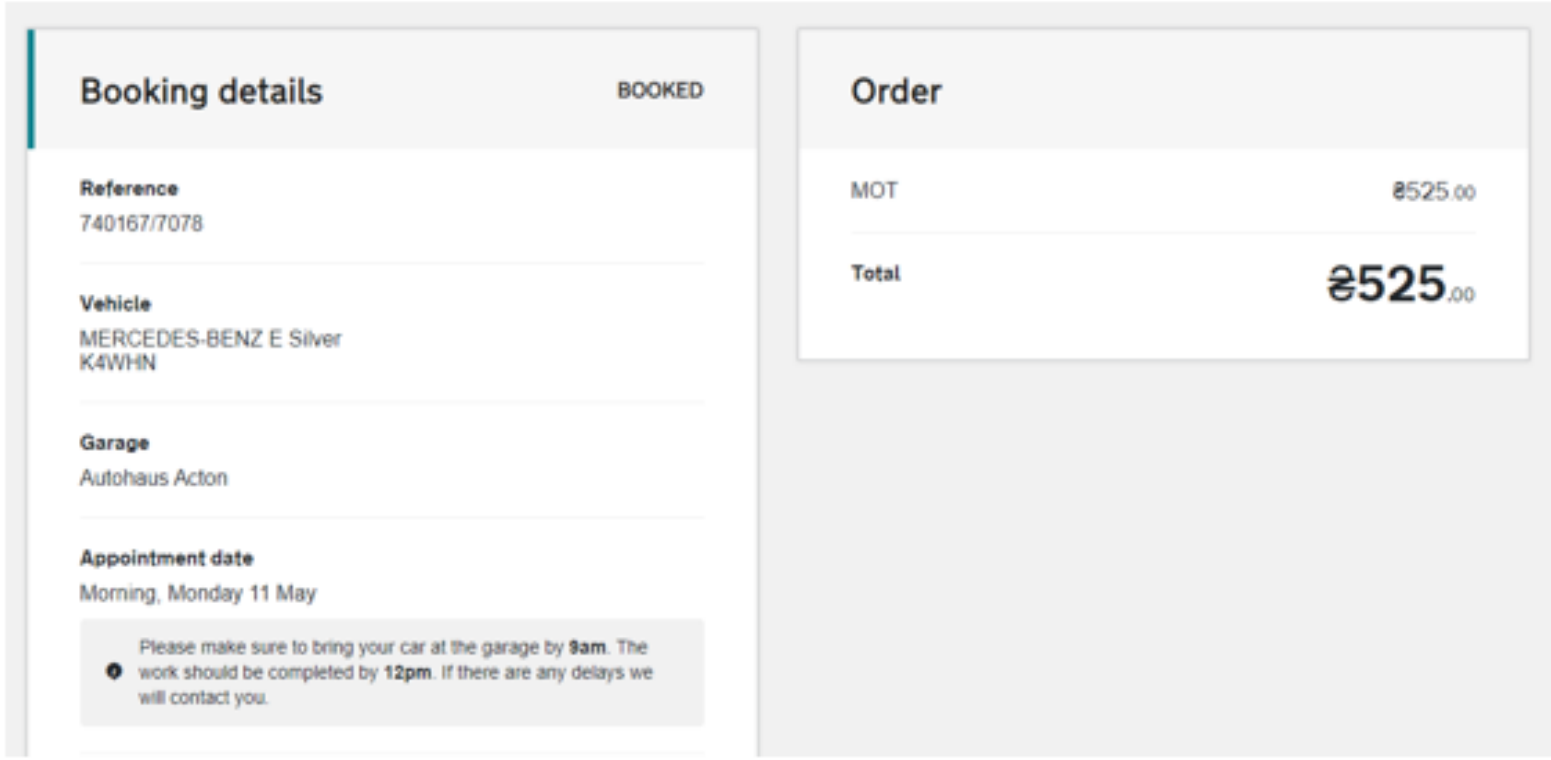
- Реєстрації авто в системі.
- Вибір сертифікованих СТО, що знаходяться поруч.
- Перегляд інформації про вільні слоти та часові рамки для виконання робіт.
- Перегляд бронювань та їх статусу.
- Оплата усіх видів робіт онлайн.
- Отримання сповіщень на електронну адресу.
- Зручний користувацький інтерфейс.
- Інтеграція компаній-партнерів із великими автопарками.
- Можливість залишати відгуки.

ПРИКЛАД РОБОТИ

ПРИКЛАД РОБОТИ



ПРИКЛАД РОБОТИ



ВИСНОВКИ

- Проведений аналіз існуючих рішень виявив потребу у створенні нового програмного продукту.
- Розроблений веб-додаток:
  - Дозволяє автоматизувати підтримку процесу технічного обслуговування автомобілів.
  - Мінімізує час очікування в чергах та додаткову комунікацію між СТО та власником транспортного засобу.
  - Має можливість інтегруватися із компаніями, що мають великі автопарки.
  - Має зрозумілий, інтуїтивний дизайн, що адаптується під пристрої із різною роздільною здатністю.
  - Надсилає сповіщення при оновленні статусу бронювання та за три дні до запланованого ТО.
- Розробку виконано в повному обсязі у відповідності до сформованих вимог.
- Тестування продукту виконано відповідно до затвердженої програми та методики тестування.

Перевірка на унікальність

UNICHECK

Submission author:
Гадиняк Руслан Анатолійович

Check ID:
1004035627

Check date:
15.06.2020 01:57:06 EEST

Check type:
Doc vs Internet + Library

Report date:
15.06.2020 09:55:11 EEST

User ID:
100000126

File name: **Телефус_основний_ПЗ**

File ID: **1004047586** Page count: **47** Word count: **9127** Character count: **67351** File size: **387.59 KB**

1.27% Matches

Highest match: **0.94%** with library source. File ID: **1000090507**

0.11% Internet Matches	1	Page 49
1.16% Library matches	16	Page 49

0% Quotes

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ ПІДТРИМКИ ПРОЦЕСУ СЕРВІСНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ілля ТЕЛЕФУС

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-додаток для підтримки процесу технічного обслуговування автомобілів. Дана система виконана у вигляді веб-додатка, написаного мовами Java та JavaScript з використанням фреймворків Spring та Angular 8 відповідно.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Доступ до бази даних через серверну частину.
2. Робота сервера із модулями, що забезпечують сповіщення на електронну адресу.
3. Робота сервера із модулем, що надає інформацію про станції ТО.
4. Обробка запитів до API компаній, що надають інформацію про свої авто.
5. Робота функціональних елементів сторінок веб-сторінки.
6. Забезпечення достатньої безпеки даних.
7. Відповідність дизайну вимогам програмного забезпечення.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1. Мануальне тестування інтерфейсу.
2. Юніт-тестування функціональності сервера.
3. Функціональне тестування точок системи.
4. E2e тестування.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність web-ресурсу перевіряється шляхом:

1. Динамічного ручного тестування – введенням невалідних значень користувачем.
2. E2e тестування веб-інтерфейсу за допомогою Puppeteer headless testing.
3. Статичного тестування коду.
4. Тестування web-ресурсу в різних web-браузерах.
5. Тестування інтерфейсу при різних роздільних здатностях екрану за допомогою Chrome developer tools.
6. Тестування при низькій швидкості інтернет з’єднання.
7. Тестування зручності використання.
8. Тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ І. ТЕЛЕФУС

2020

ЗМІСТ

1. ОПИС СТРУКТУРИ ПРОГРАМНОЇ СИСТЕМИ

Програмна система імплементована у вигляді односторінкового веб-додатку, що складається із динамічних веб-сторінок, які формуються згідно з відповіддю сервера.

Веб-додаток є одномовним, використовується англійська мова, але передбачений механізм, що дозволяє з легкістю перекладати статичний текст.

У системі існують такі статичні веб-сторінки:

1. Головна сторінка.
2. Сторінка авторизації.
3. Сторінка реєстрації.
4. Сторінка вибору категорії сервісних робіт.
5. Сторінка сервісів.
6. Сторінка ТО.
7. Сторінка позапланового ремонту.
8. Сторінка діагностики.
9. Сторінка вибору продуктів.
10. Сторінка вибору СТО у списку.
11. Сторінка вибору СТО на мапі.
12. Сторінка додавання банківської картки.
13. Сторінка вибору банківської картки.
14. Сторінка із підсумком бронювання.
15. Сторінка успішного бронювання.
16. Сторінка бронювань.
17. Сторінка окремого бронювання.
18. Сторінка оцінки бронювання.
19. Сторінка із профілем користувача.
20. Сторінка зі списком автомобілів.
21. Сторінка частих запитань.

22. Сторінка допомоги.

23. Сторінка з умовами користування.

Кожна веб-сторінка містить блок навігації (header) та елемент внизу сторінки (footer), що містять корисні посилання на усі необхідні ресурси розробленого веб-додатку.

Якщо поточний користувач є автентифікований, незалежно від статуса користувача (прямий користувач чи той, що здійснив вхід через партнерський обліковий запис), то у блоці навігації йому відображаються додаткові посилання на сторінку бронювань та для виходу із системи. Для неавтентифікованих користуавачів доступні опції входу та переходу між незахищеними сторінками. Також, незважаючи на статус авторизації користувачів – їм доступна головна сторінка із формою для вводу дани для початку процесу проходження сервісного обслуговування. Однак, авторизованих користувачів система пропустить далі, а інших перенаправить на сторінку входу, де можна авторизуватися або перейти до реєстрації.

2. ВМІСТ СТОРІНОК

Головна сторінка містить форму, що дозволяє розпочати бронювання після вводу реєстраційного номера автомобіля та поштового індекса. Також на домашній сторінці є опис переваг, які отримує користувач від використання даного веб-додатку.

Сторінка авторизації містить форму для вводу логіна та пароля, яка валідується перед відправкою даних на сервер. Якщо ж користувач ввів неправильну пару логін пароль – під формою з’явиться повідомлення про помилку, а також посилання для скидання паролю, зображене на рис. 2.1.

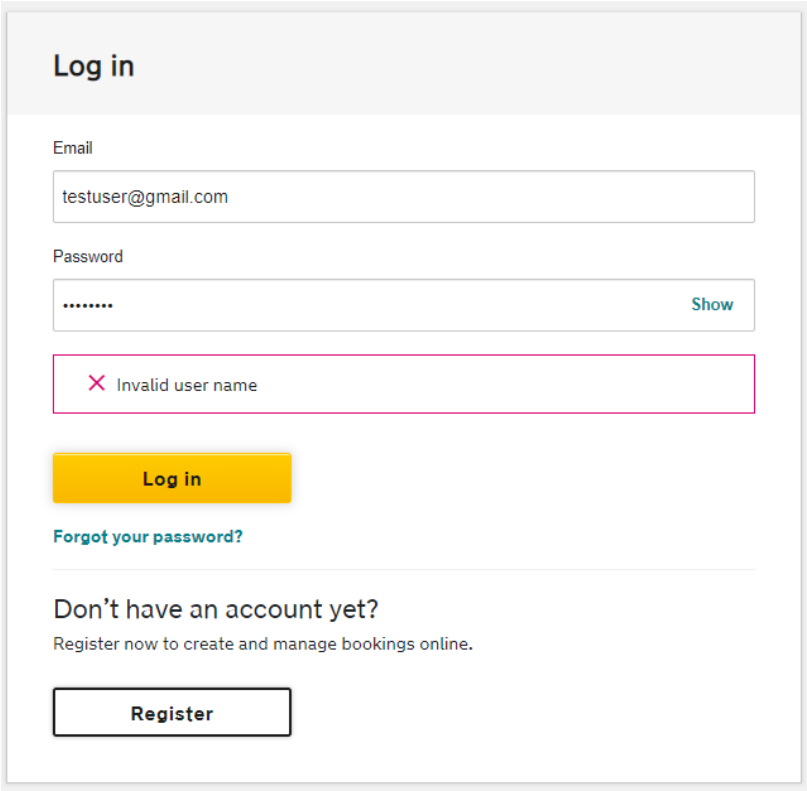


Рис. 2.1. Сторінка авторизації в системі

Після авторизації, користувач має доступ до всіх наявних сторінок. Також з'являється можливість здійснити бронювання без перенаправлення на сторінку авторизації та додати чи видалити автомобіль. Крім того, користувач може перейти на сторінку бронювань, щоб перевірити їх статус та коментарі.

Щоб додати автомобіль, користувач має перейти на сторінку Vehicles та натиснути на кнопку «Add vehicle». Після цього ввести у спливаючому дані із номерного знаку авто та натиснути кнопку «Look up» (рис. 2.2).

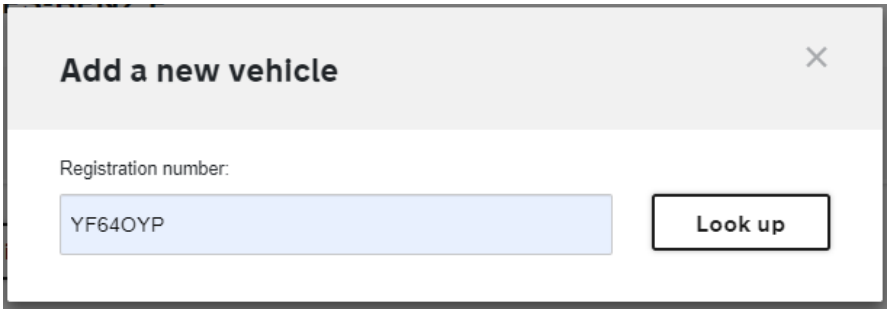


Рис. 2.2. Вікно пошуку автомобіля за номерним знаком

Після того, як автомобіль знайдено – з'являється інформація про нього та кнопка, щоб додати його у свій профіль (рис. 2.3).

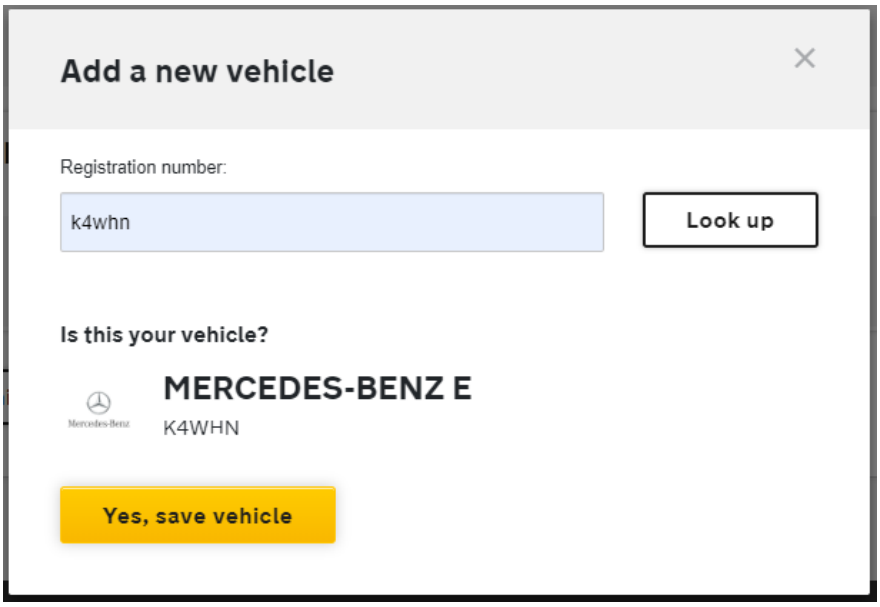


Рис. 2.3. Вікно зв’язування автомобіля з профілем користувача

Після того, як користувач додав більше одного авто – у формі бронювання при введенні номера авто зявлятиметься випадаючий список з усіма наявними транспортними засобами.

3. ПРОЦЕС АВТОРИЗАЦІЇ В СИСТЕМІ

Авторизація користувача відбувається на сторінці авторизації веб-додатку. Користувач має ввести пару «логін» та «пароль», які він вказав при реєстрації. Після введення правильних значень користувач має натиснути на кнопку «Увійти», після чого його буде перенаправлено на головну сторінку. Також якщо не авторизований користувач ввів дані авто та поштовий індекс на головній сторінці та почав бронювання – він буде перенаправлений на сторінку входу зі збереженням прогресу, а потім на попередню сторінку, на якій він був.

4. ПРОЦЕС БРОНЮВАННЯ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

Для бронювання ТО користувач має перейти на головну сторінку та ввести дані про авто і поштовий індекс (рис. 4.1).

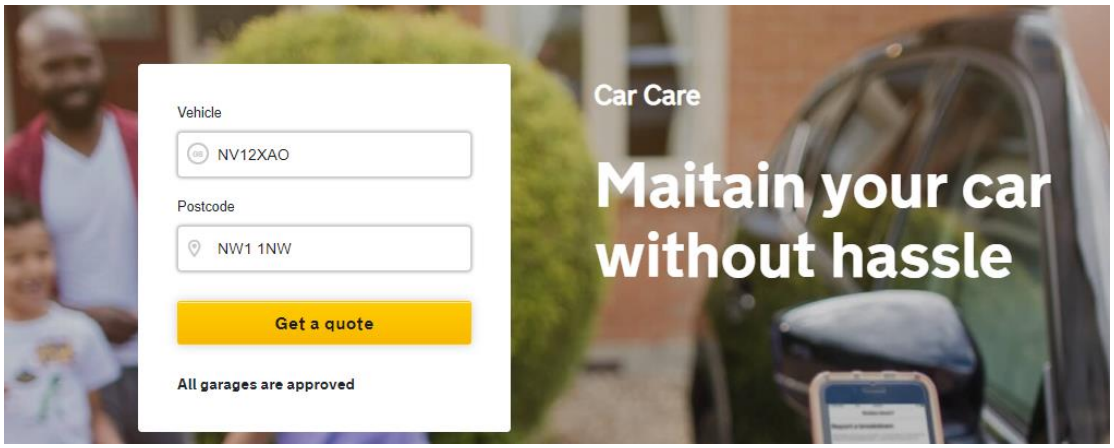


Рис. 4.1. Сторінка початку бронювань

Якщо дані, надані користувачем, правильні – здійснюється навігація на сторінку вибору категорії робіт, зображену на рис. 4.2.

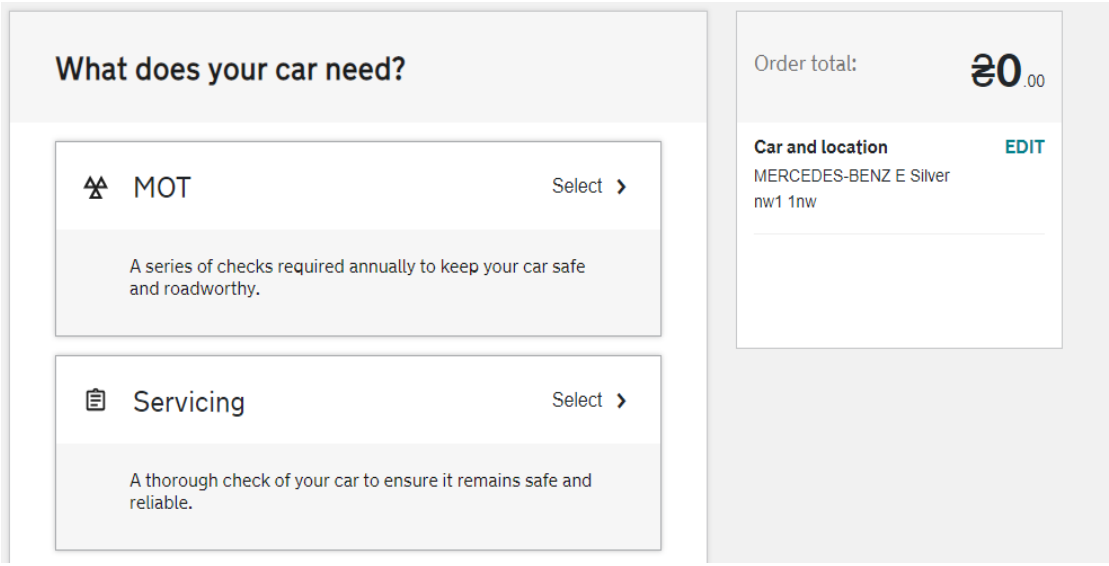


Рис. 4.2. Сторінка вибору категорії сервісних робіт

Як тільки користувач вибрав потрібний пункт, необхідно обрати конкретну послугу, після чого відбувається перенаправлення на сторінку

вибору СТО та дати візиту. Усі вільні позиції підсвічуються для більшої наглядності (рис. 4.3).

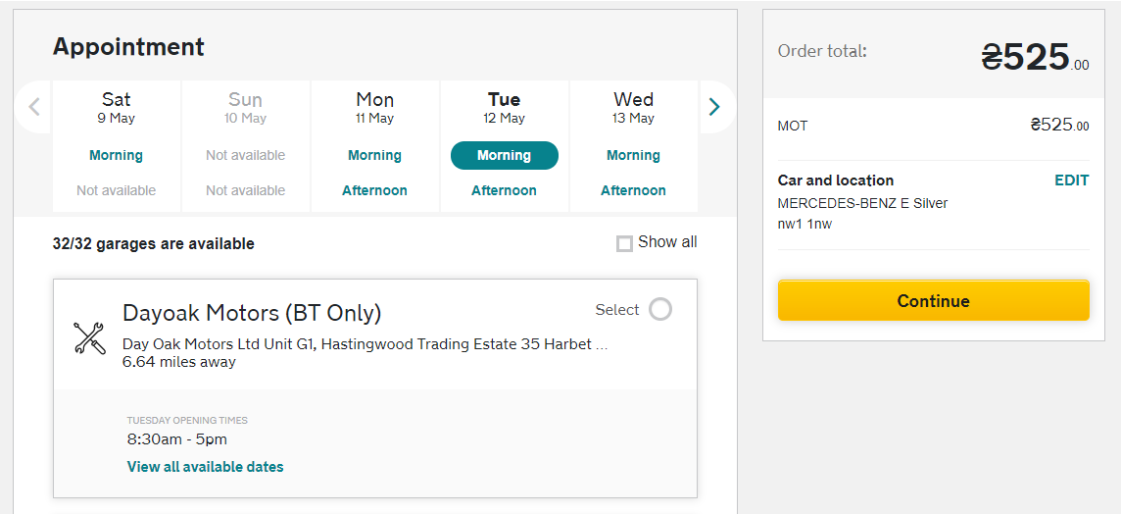


Рис. 4.3. Сторінка вибору СТО для проведення сервісних робіт

Після відбувається перехід на сторінку оплати, а далі – на сторінку із загальною інформацією про бронювання, де вказані дані водія, його автомобіля, дата та час, коли необхідно прибути для проведення ТО, а також вся інформація про сервіси, що були обрані та їх ціну. Ця інформація дублюється на сторінці My bookings (рис. 4.4).

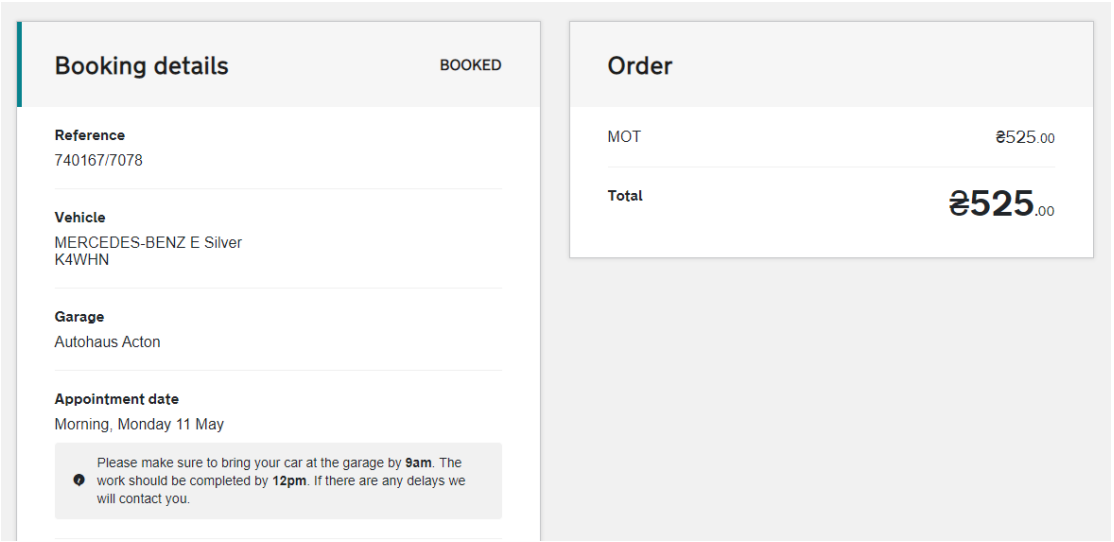


Рис. 4.4. Сторінка перегляду деталей бронювання